Exhibit C

To

Joint Claim Chart

US007310376B2

(12) **United States Patent**
Wang et al.

(10) **Patent No.: US 7,310,376 B2**
(45) **Date of Patent: Dec. 18, 2007**

(54) **MACROBLOCK LEVEL ADAPTIVE FRAME/FIELD CODING FOR DIGITAL VIDEO CONTENT**

(75) Inventors: **Limin Wang**, San Diego, CA (US); **Rajeev Gandhi**, San Diego, CA (US); **Krit Panusopone**, San Diego, CA (US); **Ajay Luthra**, San Diego, CA (US)

(73) Assignee: **General Instrument Corporation**, Horsham, PA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 100 days.

(21) Appl. No.: **11/026,394**

(22) Filed: **Dec. 30, 2004**

(65) **Prior Publication Data**

US 2005/0123043 A1 Jun. 9, 2005

Related U.S. Application Data

(62) Division of application No. 10/301,290, filed on Nov. 20, 2002, now Pat. No. 6,980,596.

(60) Provisional application No. 60/398,161, filed on Jul. 23, 2002, provisional application No. 60/395,734, filed on Jul. 12, 2002, provisional application No. 60/333,921, filed on Nov. 27, 2001.

(51) **Int. Cl.**
H04B 1/66 (2006.01)

(52) **U.S. Cl.** **375/240.24**; 375/240.25; 375/240.02; 375/240.26; 382/233; 382/235; 382/239

(58) **Field of Classification Search** 375/240.24, 375/240.25, 240.23, 240.02, 240.26; 382/233, 382/235, 239, 246, 245; 348/206
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,437,119	A	3/1984	Matsumoto et al.	
5,412,428	A	5/1995	Tahara	
5,504,530	A *	4/1996	Obikane et al.	375/240.14
5,801,778	A	9/1998	Ju	
5,825,419	A *	10/1998	Mishima et al.	375/240.15
6,094,225	A	7/2000	Han	
6,192,148	B1	2/2001	Lin	
6,404,813	B1	6/2002	Haskell et al.	

OTHER PUBLICATIONS

"Core Experiment on Interlaced Video Coding", Peter Borgwart, VideoTele.com—A Tektronix Company, Study Group 16, Question 6.

"Adaptive field/frame block coding experiment proposal", Interested Parties for the Study of Interlaced Video Coding with H.26L,Video Coding Experts Group, Study Group 16.

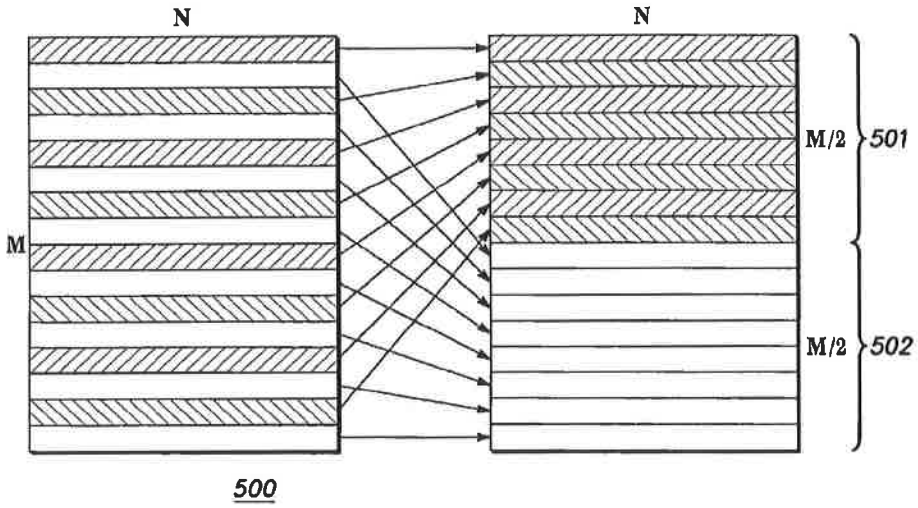
(Continued)

Primary Examiner—Shawn S. An
(74) *Attorney, Agent, or Firm*—Larry T. Cullen

(57) **ABSTRACT**

A method and system of encoding and decoding digital video content. The digital video content comprises a stream of pictures which can each be intra, predicted, or bi-predicted pictures. Each of the pictures comprises macroblocks that can be further divided into smaller blocks. The method entails encoding and decoding each of the smaller blocks in each picture in said stream of pictures in either frame mode or in field mode.

31 Claims, 8 Drawing Sheets



OTHER PUBLICATIONS

P.Borgwardt: "Core Experiment on Interlaced Video Coding VCEG-N85" ITU-Telecommunications Standardization Sector ITU-T Q.6/SG16 Video Coding Expert Group (VCEG) 24-27 Sep. 2001, pp. 1-10, XP002257037 Santa Barbara, CA USA.

"H.26L Test Model Long Term Number 8 (TML-8) Drafto" ITU-T Telecommunication Standardization Sector of ITU, Geneva, CH, 2 Apr. 2001 (Apr. 2, 2001), pp.1-54, XP001089814 p. 9, paragraph 1 - p. 10, paragraph 6.1.2.2.1 p p. 40.

P. Borgwardt: "Handling Interlaced Video in H.26l VCEG-N57" ITU-Telecommunications Standardization Sector ITU Q.6/SG16 Video Coding Expert Group (VCEG), 24-27 Sep. 2001, pp. 1-3, XP002257142 Santa Barbara, CA USA.

M. Gallant et al: High Rate, High Resolution Video Using H.26L VCEG-N84 ITU-Telecommunications Standarization Secotr ITU Q.6/SG16 Video Coding Expert Group (VCEG), 24-27 Sep. 2001, pp. 1-7, XP002257143 Santa Barbara, CA USA; p. 1; p. 3.

"Adaptive Field/From Block Coding Experiment Proposal". Interested Parties for the Study of Interfaced Video Coding with H.26L, Video Coding Experts Group, Study Group 16.

* cited by examiner

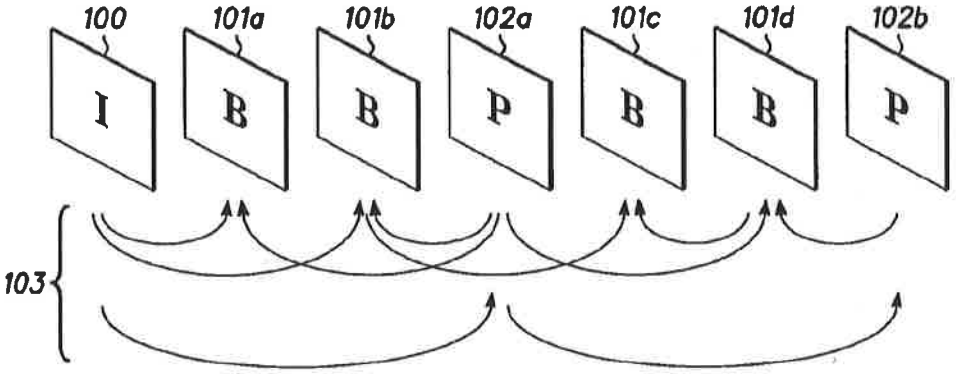


FIG. 1

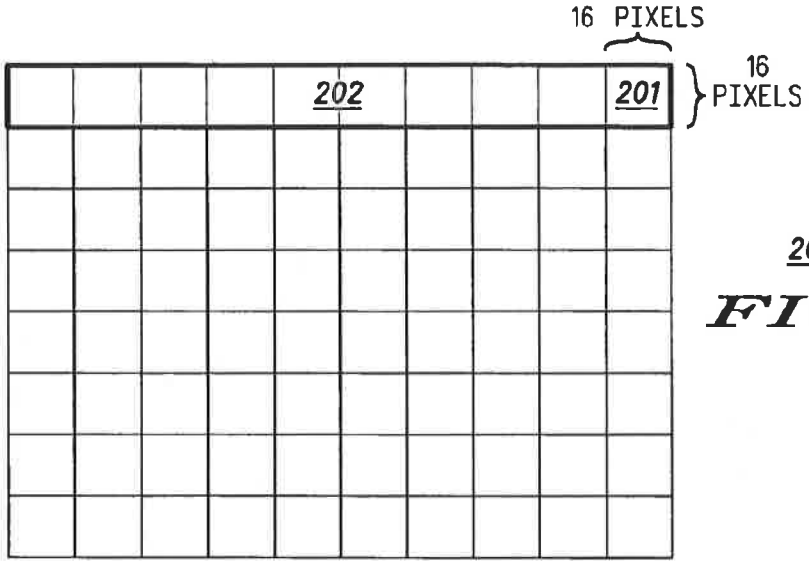


FIG. 2

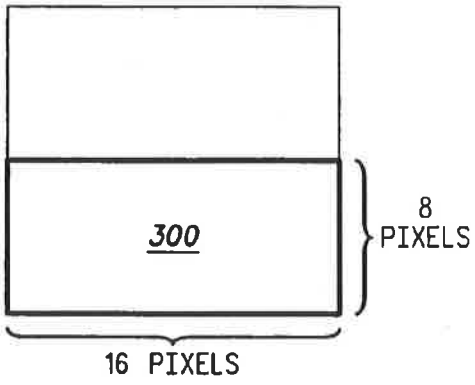


FIG. 3A

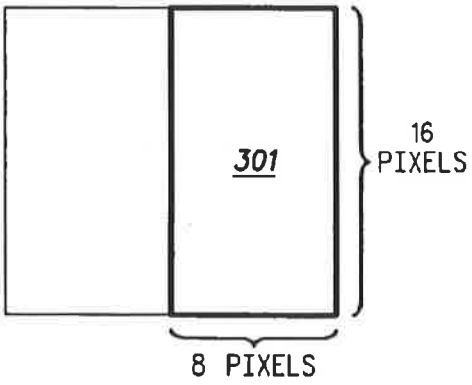


FIG. 3B

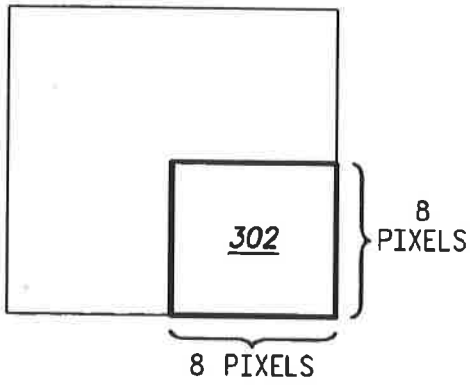


FIG. 3C

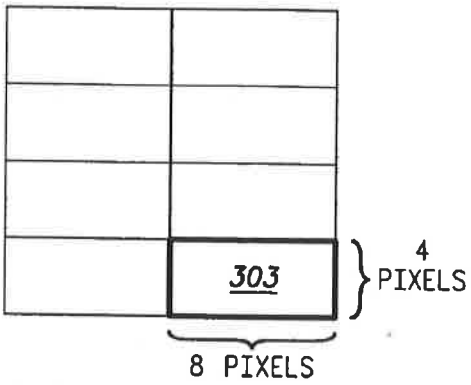


FIG. 3D

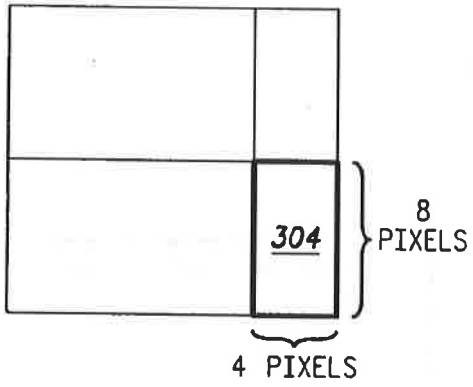


FIG. 3E

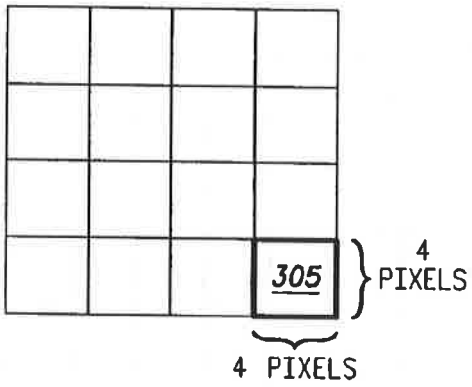


FIG. 3F

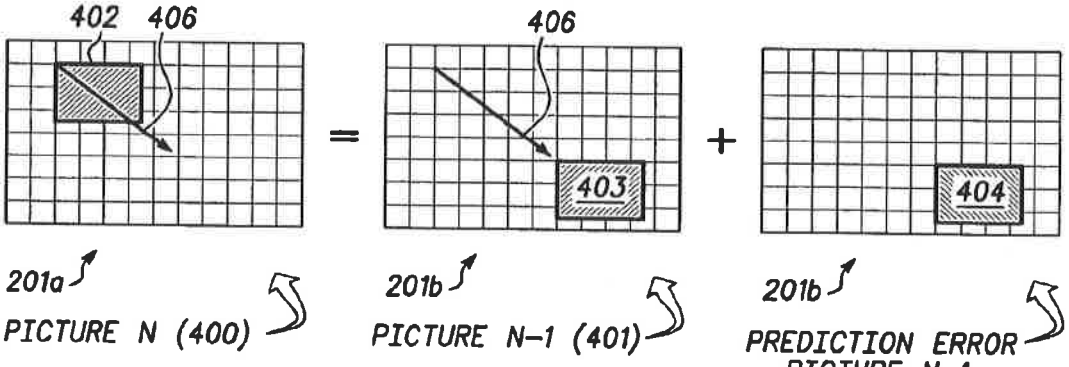


FIG. 4

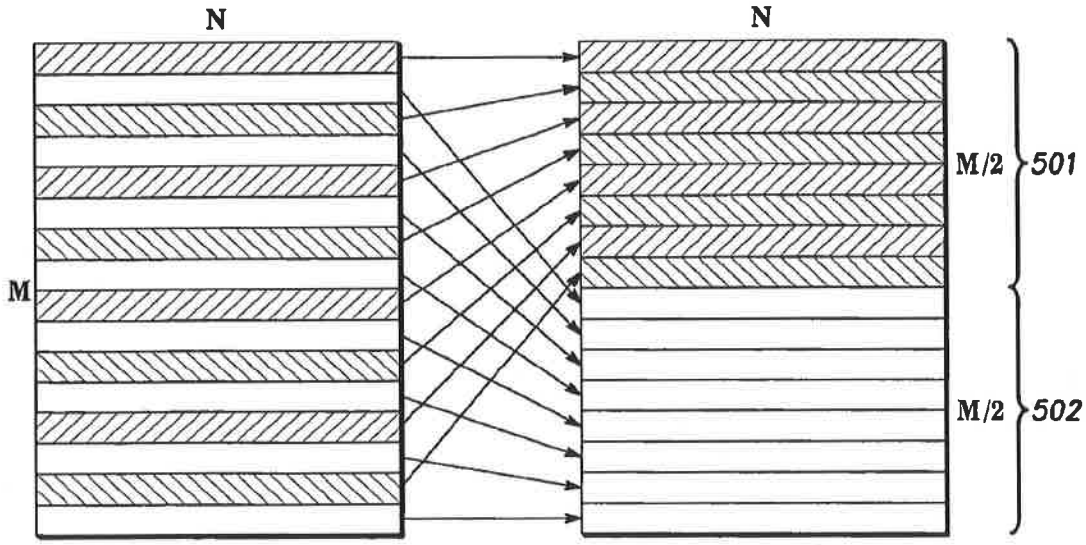


FIG. 5

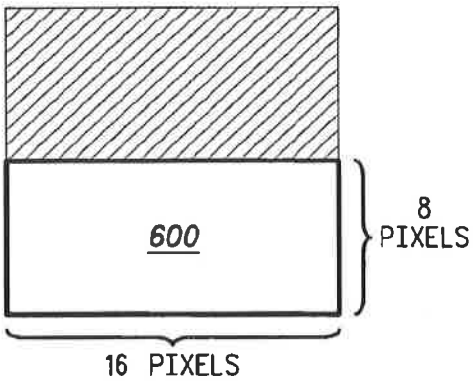


FIG. 6A

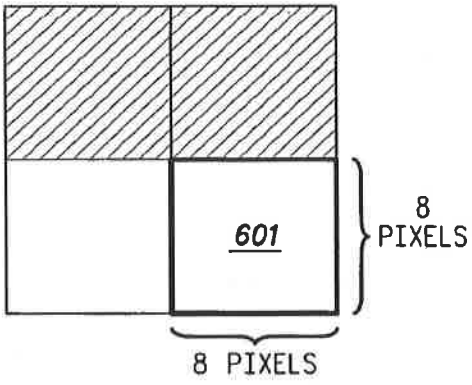


FIG. 6B

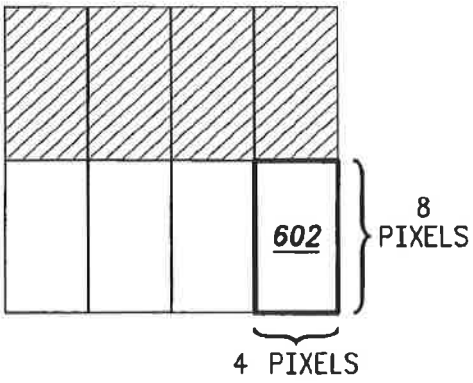


FIG. 6C

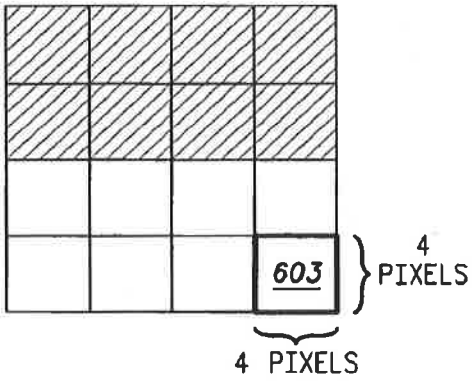


FIG. 6D

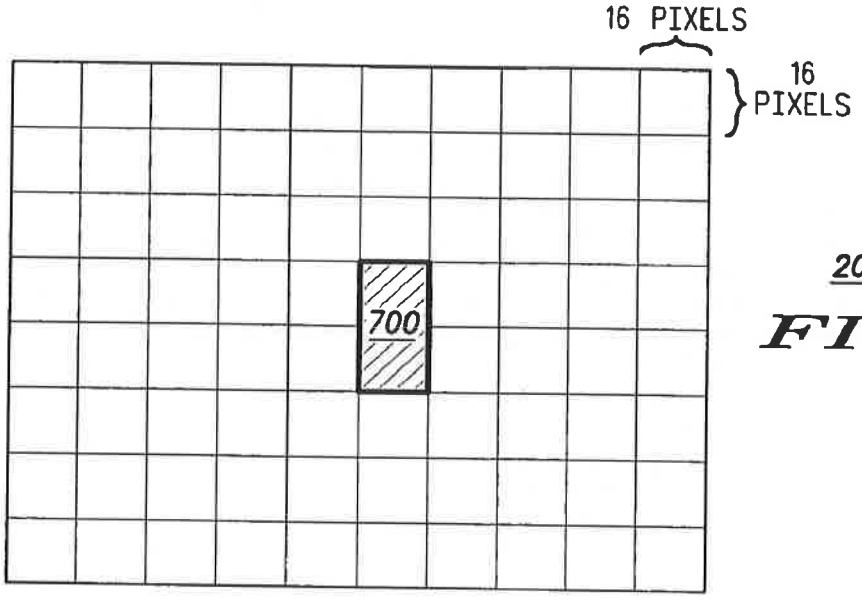


FIG. 7

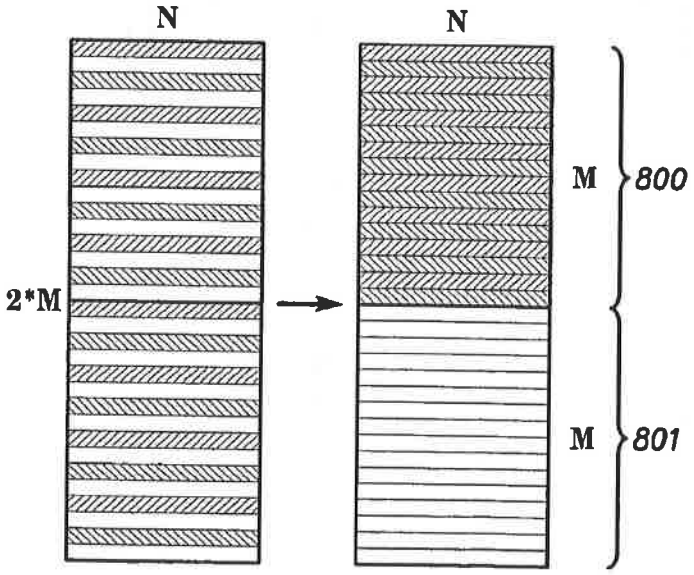


FIG. 8

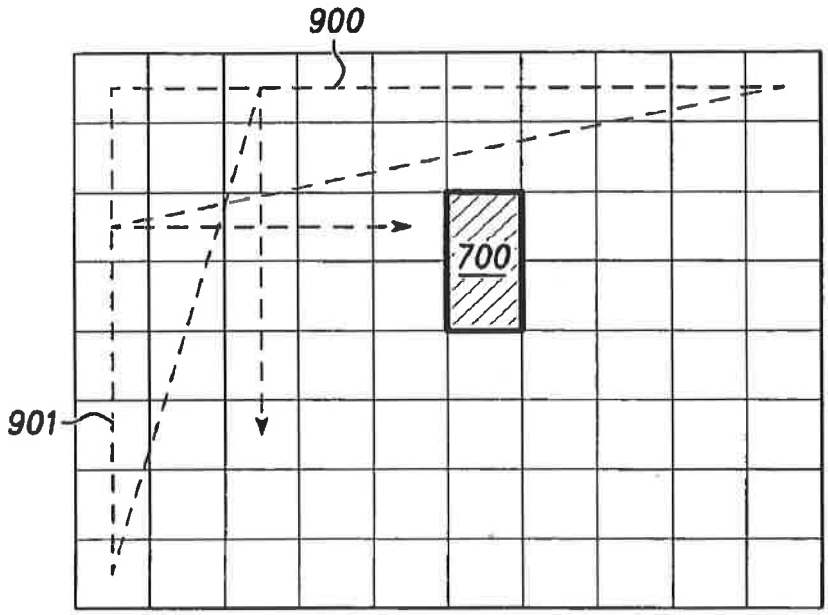


FIG. 9

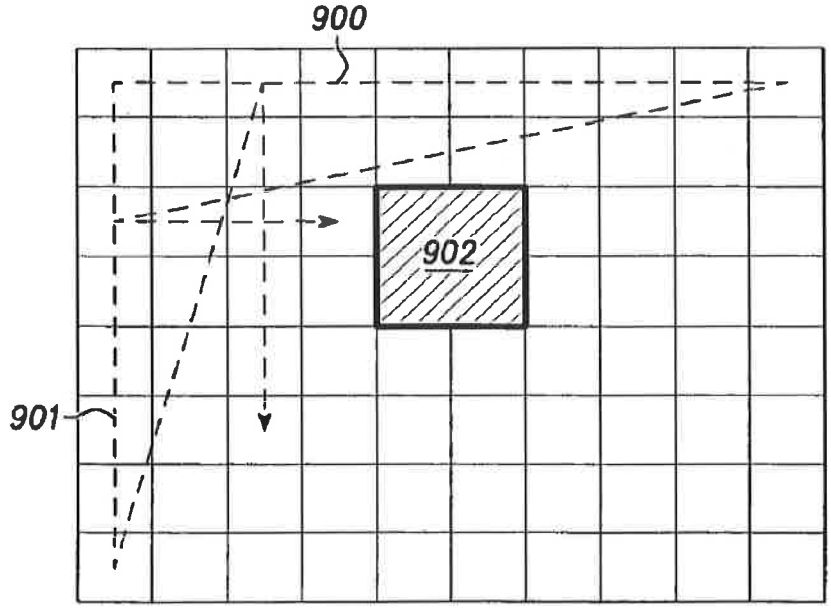


FIG. 10

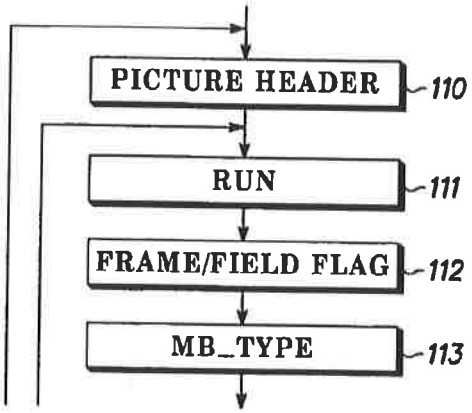


FIG. 11

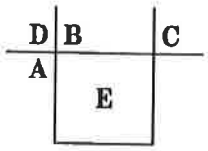


FIG. 12

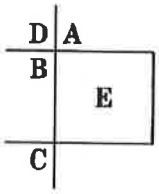


FIG. 13

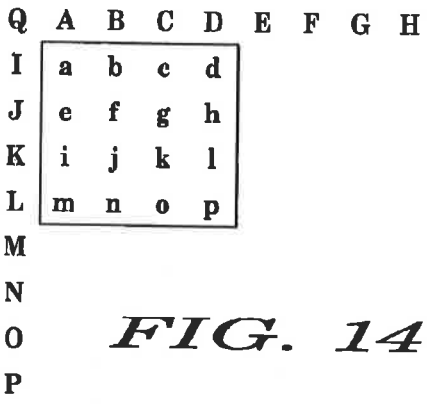


FIG. 14

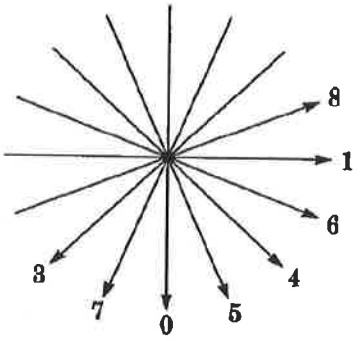


FIG. 15

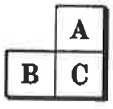


FIG. 16A

0	1	4	5
2	3	6	7
8	9	12	13
10	11	14	15

FIG. 16B

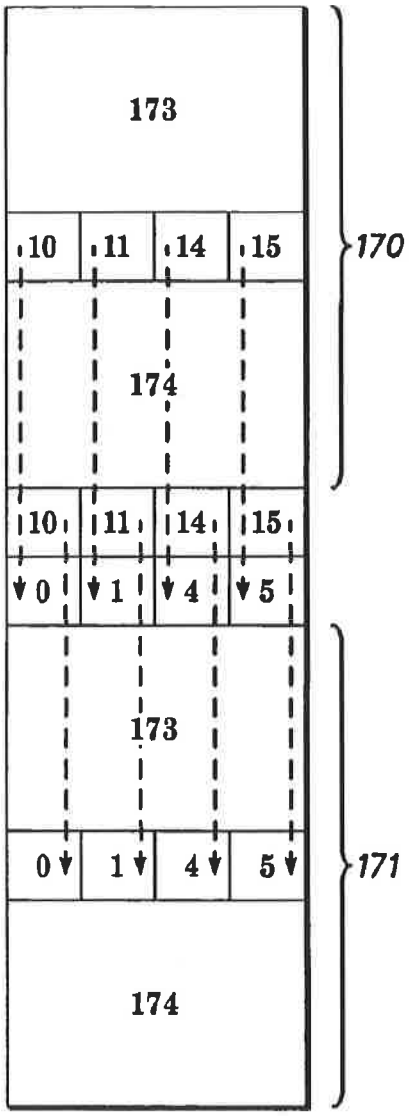


FIG. 17A

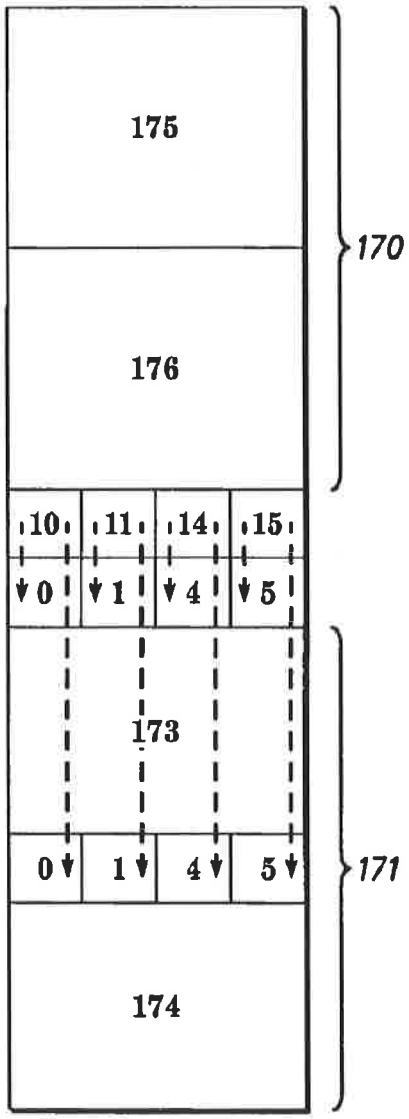


FIG. 17B

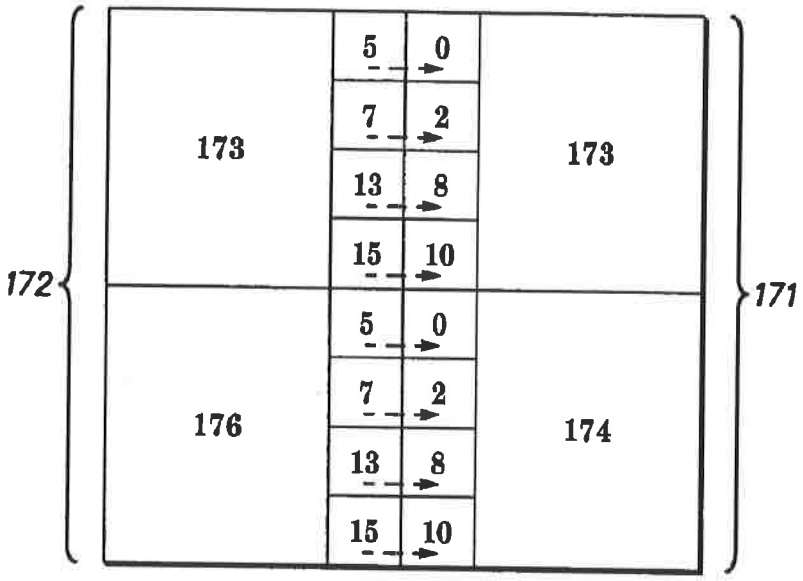


FIG. 17C

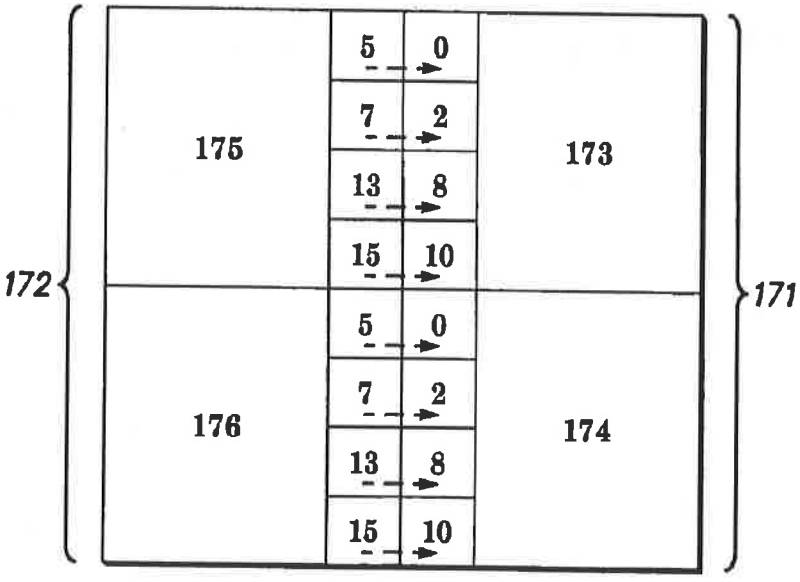


FIG. 17D

MACROBLOCK LEVEL ADAPTIVE FRAME/FIELD CODING FOR DIGITAL VIDEO CONTENT

The present application claims priority under 35 U.S.C. §119(e) from the following previously filed Provisional patent applications: Ser. No. 60/333,921, filed Nov. 27, 2001; Ser. No. 60/395,734, filed Jul. 12, 2002; Ser. No. 60/398,161, filed Jul. 23, 2002; all of which are herein incorporated by reference. This application is also a Divisional of U.S. patent application Ser. No. 10/301,290 filed on Nov. 20, 2002 now U.S. Pat. No. 6,980,596, which is herein incorporated by reference.

TECHNICAL FIELD

The present invention relates to encoding and decoding of digital video content. More specifically, the present invention relates to frame mode and field mode encoding of digital video content at a macroblock level as used in the MPEG-4 Part 10 AVC/H.264 standard video coding standard.

BACKGROUND

Video compression is used in many current and emerging products. It is at the heart of digital television set-top boxes (STBs), digital satellite systems (DSSs), high definition television (HDTV) decoders, digital versatile disk (DVD) players, video conferencing, Internet video and multimedia content, and other digital video applications. Without video compression, digital video content can be extremely large, making it difficult or even impossible for the digital video content to be efficiently stored, transmitted, or viewed.

The digital video content comprises a stream of pictures that can be displayed as an image on a television receiver, computer monitor, or some other electronic device capable of displaying digital video content. A picture that is displayed in time before a particular picture is in the "backward direction" in relation to the particular picture. Likewise, a picture that is displayed in time after a particular picture is in the "forward direction" in relation to the particular picture.

Video compression is accomplished in a video encoding, or coding, process in which each picture is encoded as either a frame or as two fields. Each frame comprises a number of lines of spatial information. For example, a typical frame contains 480 horizontal lines. Each field contains half the number of lines in the frame. For example, if the frame comprises 480 horizontal lines, each field comprises 240 horizontal lines. In a typical configuration, one of the fields comprises the odd numbered lines in the frame and the other field comprises the even numbered lines in the frame. The field that comprises the odd numbered lines will be referred to as the "top" field hereafter and in the appended claims, unless otherwise specifically denoted. Likewise, the field that comprises the even numbered lines will be referred to as the "bottom" field hereafter and in the appended claims, unless otherwise specifically denoted. The two fields can be interlaced together to form an interlaced frame.

The general idea behind video coding is to remove data from the digital video content that is "non-essential." The decreased amount of data then requires less bandwidth for broadcast or transmission. After the compressed video data has been transmitted, it must be decoded, or decompressed. In this process, the transmitted video data is processed to generate approximation data that is substituted into the video data to replace the "non-essential" data that was removed in the coding process.

Video coding transforms the digital video content into a compressed form that can be stored using less space and transmitted using less bandwidth than uncompressed digital video content. It does so by taking advantage of temporal and spatial redundancies in the pictures of the video content. The digital video content can be stored in a storage medium such as a hard drive, DVD, or some other non-volatile storage unit.

There are numerous video coding methods that compress the digital video content. Consequently, video coding standards have been developed to standardize the various video coding methods so that the compressed digital video content is rendered in formats that a majority of video encoders and decoders can recognize. For example, the Motion Picture Experts Group (MPEG) and International Telecommunication Union (ITU-T) have developed video coding standards that are in wide use. Examples of these standards include the MPEG-1, MPEG-2, MPEG-4, ITU-T H261, and ITU-T H263 standards.

Most modern video coding standards, such as those developed by MPEG and ITU-T, are based in part on a temporal prediction with motion compensation (MC) algorithm. Temporal prediction with motion compensation is used to remove temporal redundancy between successive pictures in a digital video broadcast.

The temporal prediction with motion compensation algorithm typically utilizes one or two reference pictures to encode a particular picture. A reference picture is a picture that has already been encoded. By comparing the particular picture that is to be encoded with one of the reference pictures, the temporal prediction with motion compensation algorithm can take advantage of the temporal redundancy that exists between the reference picture and the particular picture that is to be encoded and encode the picture with a higher amount of compression than if the picture were encoded without using the temporal prediction with motion compensation algorithm. One of the reference pictures may be in the backward direction in relation to the particular picture that is to be encoded. The other reference picture is in the forward direction in relation to the particular picture that is to be encoded.

However, as the demand for higher resolutions, more complex graphical content, and faster transmission time increases, so does the need for better video compression methods. To this end, a new video coding standard is currently being developed jointly by ISO and ITU-T. This new video coding standard is called the MPEG-4 Advanced Video Coding (AVC)/H.264 standard.

SUMMARY OF THE INVENTION

In one of many possible embodiments, the present invention provides a method of encoding, decoding, and bitstream generation of digital video content. The digital video content comprises a stream of pictures which can each be intra, predicted, or bi-predicted pictures. Each of the pictures comprises macroblocks that can be further divided into smaller blocks. The method entails encoding and decoding each of the macroblocks in each picture in said stream of pictures in either frame mode or in field mode.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings illustrate various embodiments of the present invention and are a part of the specification. Together with the following description, the drawings demonstrate and explain the principles of the present

invention. The illustrated embodiments are examples of the present invention and do not limit the scope of the invention.

FIG. **1** illustrates an exemplary sequence of three types of pictures that can be used to implement the present invention, as defined by an exemplary video coding standard such as the MPEG-4 Part 10 AVC/H.264 standard.

FIG. **2** shows that each picture is preferably divided into slices containing macroblocks according to an embodiment of the present invention.

FIG. **3***a* shows that a macroblock can be further divided into a block size of 16 by 8 pixels according to an embodiment of the present invention.

FIG. **3***b* shows that a macroblock can be further divided into a block size of 8 by 16 pixels according to an embodiment of the present invention.

FIG. **3***c* shows that a macroblock can be further divided into a block size of 8 by 8 pixels according to an embodiment of the present invention.

FIG. **3***d* shows that a macroblock can be further divided into a block size of 8 by 4 pixels according to an embodiment of the present invention.

FIG. **3***e* shows that a macroblock can be further divided into a block size of 4 by 8 pixels according to an embodiment of the present invention.

FIG. **3***f* shows that a macroblock can be further divided into a block size of 4 by 4 pixels according to an embodiment of the present invention.

FIG. **4** shows a picture construction example using temporal prediction with motion compensation that illustrates an embodiment of the present invention.

FIG. **5** shows that a macroblock is split into a top field and a bottom field if it is to be encoded in field mode.

FIG. **6***a* shows that a macroblock that is encoded in field mode can be divided into a block with a size of 16 by 8 pixels according to an embodiment of the present invention.

FIG. **6***b* shows that a macroblock that is encoded in field mode can be divided into a block with a size of 8 by 8 pixels according to an embodiment of the present invention.

FIG. **6***c* shows that a macroblock that is encoded in field mode can be divided into a block with a size of 4 by 8 pixels according to an embodiment of the present invention.

FIG. **6***d* shows that a macroblock that is encoded in field mode can be divided into a block with a size of 4 by 4 pixels according to an embodiment of the present invention.

FIG. **7** illustrates an exemplary pair of macroblocks that can be used in AFF coding on a pair of macroblocks according to an embodiment of the present invention.

FIG. **8** shows that a pair of macroblocks that is to be encoded in field mode is first split into one top field 16 by 16 pixel block and one bottom field 16 by 16 pixel block.

FIG. **9** shows two possible scanning paths in AFF coding of pairs of macroblocks.

FIG. **10** illustrates another embodiment of the present invention which extends the concept of AFF coding on a pair of macroblocks to AFF coding to a group of four or more neighboring macroblocks.

FIG. **11** shows some of the information included in the bitstream which contains information pertinent to each macroblock within a stream.

FIG. **12** shows a block that is to be encoded and its neighboring blocks and will be used to explain various preferable methods of calculating the PMV of a block in a macroblock.

FIG. **13** shows an alternate definition of neighboring blocks if the scanning path is a vertical scanning path.

FIG. **14** shows that each pixel value is predicted from neighboring blocks' pixel values according to an embodiment of the present invention.

FIG. **15** shows different prediction directions for intra_4×4 coding.

FIGS. **16***a-b* illustrate that the chosen intra-prediction mode (intra_pred_mode) of a 4 by 4 pixel block is highly correlated with the prediction modes of adjacent blocks.

FIGS. **17***a-d* show neighboring blocks definitions in relation to a current macroblock pair that is to be encoded.

Throughout the drawings, identical reference numbers designate similar, but not necessarily identical, elements.

DETAILED DESCRIPTION OF EMBODIMENTS OF THE INVENTION

The present invention provides a method of adaptive frame/field (AFF) coding of digital video content comprising a stream of pictures or slices of a picture at a macroblock level. The present invention extends the concept of picture level AFF to macroblocks. In AFF coding at a picture level, each picture in a stream of pictures that is to be encoded is encoded in either frame mode or in field mode, regardless of the frame or field coding mode of other pictures that are to be coded. If a picture is encoded in frame mode, the two fields that make up an interlaced frame are coded jointly. Conversely, if a picture is encoded in field mode, the two fields that make up an interlaced frame are coded separately. The encoder determines which type of coding, frame mode coding or field mode coding, is more advantageous for each picture and chooses that type of encoding for the picture. The exact method of choosing between frame mode and field mode is not critical to the present invention and will not be detailed herein.

As noted above, the MPEG-4 Part 10 AVC/H.264 standard is a new standard for encoding and compressing digital video content. The documents establishing the MPEG-4 Part 10 AVC/H.264 standard are hereby incorporated by reference, including "Joint Final Committee Draft (JFCD) of Joint Video Specification" issued by the Joint Video Team (JVT) on Aug. 10, 2002. (ITU-T Rec. H.264 & ISO/IEC 14496-10 AVC). The JVT consists of experts from ISO or MPEG and ITU-T. Due to the public nature of the MPEG-4 Part 10 AVC/H.264 standard, the present specification will not attempt to document all the existing aspects of MPEG-4 Part 10 AVC/H.264 video coding, relying instead on the incorporated specifications of the standard.

Although this method of AFF encoding is compatible with and will be explained using the MPEG-4 Part 10 AVC/H.264 standard guidelines, it can be modified and used as best serves a particular standard or application.

Using the drawings, the preferred embodiments of the present invention will now be explained.

FIG. **1** illustrates an exemplary sequence of three types of pictures that can be used to implement the present invention, as defined by an exemplary video coding standard such as the MPEG-4 Part 10 AVC/H.264 standard. As previously mentioned, the encoder encodes the pictures and the decoder decodes the pictures. The encoder or decoder can be a processor, application specific integrated circuit (ASIC), field programmable gate array (FPGA), coder/decoder (CODEC), digital signal processor (DSP), or some other electronic device that is capable of encoding the stream of pictures. However, as used hereafter and in the appended claims, unless otherwise specifically denoted, the term "encoder" will be used to refer expansively to all electronic devices that encode digital video content comprising a

stream of pictures. The term "decoder" will be used to refer expansively to all electronic devices that decode digital video content comprising a stream of pictures.

As shown in FIG. **1**, there are preferably three types of pictures that can be used in the video coding method. Three types of pictures are defined to support random access to stored digital video content while exploring the maximum redundancy reduction using temporal prediction with motion compensation. The three types of pictures are intra (I) pictures (**100**), predicted (P) pictures (**102***a,b*), and bi-predicted (B) pictures (**101***a-d*). An I picture (**100**) provides an access point for random access to stored digital video content and can be encoded only with slight compression. Intra pictures (**100**) are encoded without referring to reference pictures.

A predicted picture (**102***a,b*) is encoded using an I, P, or B picture that has already been encoded as a reference picture. The reference picture can be in either the forward or backward temporal direction in relation to the P picture that is being encoded. The predicted pictures (**102***a,b*) can be encoded with more compression than the intra pictures (**100**).

A bi-predicted picture (**101***a-d*) is encoded using two temporal reference pictures: a forward reference picture and a backward reference picture. The forward reference picture is sometimes called a past reference picture and the backward reference picture is sometimes called a future reference picture. An embodiment of the present invention is that the forward reference picture and backward reference picture can be in the same temporal direction in relation to the B picture that is being encoded. Bi-predicted pictures (**101***a-d*) can be encoded with the most compression out of the three picture types.

Reference relationships (**103**) between the three picture types are illustrated in FIG. **1**. For example, the P picture (**102***a*) can be encoded using the encoded I picture (**100**) as its reference picture. The B pictures (**101***a-d*) can be encoded using the encoded I picture (**100**) or the encoded P picture (**102***a*) as its reference pictures, as shown in FIG. **1**. Under the principles of an embodiment of the present invention, encoded B pictures (**101***a-d*) can also be used as reference pictures for other B pictures that are to be encoded. For example, the B picture (**101***c*) of FIG. **1** is shown with two other B pictures (**101***b* and **101***d*) as its reference pictures.

The number and particular order of the I (**100**), B (**101***a-d*), and P (**102***a,b*) pictures shown in FIG. **1** are given as an exemplary configuration of pictures, but are not necessary to implement the present invention. Any number of I, B, and P pictures can be used in any order to best serve a particular application. The MPEG-4 Part 10 AVC/H.264 standard does not impose any limit to the number of B pictures between two reference pictures nor does it limit the number of pictures between two I pictures.

FIG. **2** shows that each picture (**200**) is preferably divided into slices (**202**). A slice (**202**) comprises a group of macroblocks (**201**). A macroblock (**201**) is a rectangular group of pixels. As shown in FIG. **2**, a preferable macroblock (**201**) size is 16 by 16 pixels.

FIGS. **3***a-f* show that a macroblock can be further divided into smaller sized blocks. For example, as shown in FIGS. **3***a-f*, a macroblock can be further divided into block sizes of 16 by 8 pixels (FIG. **3***a*; **300**), 8 by 16 pixels (FIG. **3***b*; **301**), 8 by 8 pixels (FIG. **3***c*; **302**), 8 by 4 pixels (FIG. **3***d*; **303**), 4 by 8 pixels (FIG. **3***e*; **304**), or 4 by 4 pixels (FIG. **3***f*; **305**). These smaller block sizes are preferable in some applications that use the temporal prediction with motion compensation algorithm.

FIG. **4** shows a picture construction example using temporal prediction with motion compensation that illustrates an embodiment of the present invention. Temporal prediction with motion compensation assumes that a current picture, picture N (**400**), can be locally modeled as a translation of another picture, picture N-1 (**401**). The picture N-1 (**401**) is the reference picture for the encoding of picture N (**400**) and can be in the forward or backwards temporal direction in relation to picture N (**400**).

As shown in FIG. **4**, each picture is preferably divided into slices containing macroblocks (**201***a,b*). The picture N-1 (**401**) contains an image (**403**) that is to be shown in picture N (**400**). The image (**403**) will be in a different temporal position in picture N (**402**) than it is in picture N-1 (**401**), as shown in FIG. **4**. The image content of each macroblock (**201***b*) of picture N (**400**) is predicted from the image content of each corresponding macroblock (**201***a*) of picture N-1 (**401**) by estimating the required amount of temporal motion of the image content of each macroblock (**201***a*) of picture N-1 (**401**) for the image (**403**) to move to its new temporal position (**402**) in picture N (**400**). Instead of the original image (**402**) being encoded, the difference (**404**) between the image (**402**) and its prediction (**403**) is actually encoded and transmitted.

For each image (**402**) in picture N (**400**), the temporal prediction can often be described by motion vectors that represent the amount of temporal motion required for the image (**403**) to move to a new temporal position in the picture N (**402**). The motion vectors (**406**) used for the temporal prediction with motion compensation need to be encoded and transmitted.

FIG. **4** shows that the image (**402**) in picture N (**400**) can be represented by the difference (**404**) between the image and its prediction and the associated motion vectors (**406**). The exact method of encoding using the motion vectors can vary as best serves a particular application and can be easily implemented by someone who is skilled in the art.

To understand macroblock level AFF coding, a brief overview of picture level AFF coding of a stream of pictures will now be given. A frame of an interlaced sequence contains two fields, the top field and the bottom field, which are interleaved and separated in time by a field period. The field period is half the time of a frame period. In picture level AFF coding, the two fields of an interlaced frame can be coded jointly or separately. If they are coded jointly, frame mode coding is used. Conversely, if the two fields are coded separately, field mode coding is used.

Fixed frame/field coding, on the other hand, codes all the pictures in a stream of pictures in one mode only. That mode can be frame mode or it can be field mode. Picture level AFF is preferable to fixed frame/field coding in many applications because it allows the encoder to chose which mode, frame mode or field mode, to encode each picture in the stream of pictures based on the contents of the digital video material. AFF coding results in better compression than does fixed frame/field coding in many applications.

An embodiment of the present invention is that AFF coding can be performed on smaller portions of a picture. This small portion can be a macroblock, a pair of macroblocks, or a group of macroblocks. Each macroblock, pair of macroblocks, or group of macroblocks or slice is encoded in frame mode or in field mode, regardless of how the other macroblocks in the picture are encoded. AFF coding in each of the three cases will be described in detail below.

In the first case, AFF coding is performed on a single macroblock. If the macroblock is to be encoded in frame mode, the two fields in the macroblock are encoded jointly. Once encoded as a frame, the macroblock can be further divided into the smaller blocks of FIGS. **3***a-f* for use in the temporal prediction with motion compensation algorithm.

However, if the macroblock is to be encoded in field mode, the macroblock (**500**) is split into a top field (**501**) and a bottom field (**502**), as shown in FIG. **5**. The two fields are then coded separately. In FIG. **5**, the macroblock has M rows of pixels and N columns of pixels. A preferable value of N and M is 16, making the macroblock (**500**) a 16 by 16 pixel macroblock. As shown in FIG. **5**, every other row of pixels is shaded. The shaded areas represent the rows of pixels in the top field of the macroblock (**500**) and the unshaded areas represent the rows of pixels in the bottom field of the macroblock (**500**).

As shown in FIGS. **6***a-d*, a macroblock that is encoded in field mode can be divided into four additional blocks. A block is required to have a single parity. The single parity requirement is that a block cannot comprise both top and bottom fields. Rather, it must contain a single parity of field. Thus, as shown in FIGS. **6***a-d*, a field mode macroblock can be divided into blocks of 16 by 8 pixels (FIG. **6***a*; **600**), 8 by 8 pixels (FIG. **6***b*; **601**), 4 by 8 pixels (FIG. **6***c*; **602**), and 4 by 4 pixels (FIG. **6***d*; **603**). FIGS. **6***a-d* shows that each block contains fields of a single parity.

AFF coding on macroblock pairs will now be explained. AFF coding on macroblock pairs will be occasionally referred to as pair based AFF coding. A comparison of the block sizes in FIGS. **6***a-d* and in FIGS. **3***a-f* show that a macroblock encoded in field mode can be divided into fewer block patterns than can a macroblock encoded in frame mode. The block sizes of 16 by 16 pixels, 8 by 16 pixels, and 8 by 4 pixels are not available for a macroblock encoded in field mode because of the single parity requirement. This implies that the performance of single macroblock based AFF may not be good for some sequences or applications that strongly favor field mode coding. In order to guarantee the performance of field mode macroblock coding, it is preferable in some applications for macroblocks that are coded in field mode to have the same block sizes as macroblocks that are coded in frame mode. This can be achieved by performing AFF coding on macroblock pairs instead of on single macroblocks.

FIG. **7** illustrates an exemplary pair of macroblocks (**700**) that can be used in AFF coding on a pair of macroblocks according to an embodiment of the present invention. If the pair of macroblocks (**700**) is to be encoded in frame mode, the pair is coded as two frame-based macroblocks. In each macroblock, the two fields in each of the macroblocks are encoded jointly. Once encoded as frames, the macroblocks can be further divided into the smaller blocks of FIGS. **3***a-f* for use in the temporal prediction with motion compensation algorithm.

However, if the pair of macroblocks (**700**) is to be encoded in field mode, it is first split into one top field 16 by 16 pixel block (**800**) and one bottom field 16 by 16 pixel block (**801**), as shown in FIG. **8**. The two fields are then coded separately. In FIG. **8**, each macroblock in the pair of macroblocks (**700**) has N=16 columns of pixels and M=16 rows of pixels. Thus, the dimensions of the pair of macroblocks (**700**) is 16 by 32 pixels. As shown in FIG. **8**, every other row of pixels is shaded. The shaded areas represent the rows of pixels in the top field of the macroblocks and the unshaded areas represent the rows of pixels in the bottom field of the macroblocks. The top field block (**800**) and the bottom field block (**801**) can now be divided into one of the possible block sizes of FIGS. **3***a-f*.

According to an embodiment of the present invention, in the AFF coding of pairs of macroblocks (**700**), there are two possible scanning paths. A scanning path determines the order in which the pairs of macroblocks of a picture are encoded. FIG. **9** shows the two possible scanning paths in AFF coding of pairs of macroblocks (**700**). One of the scanning paths is a horizontal scanning path (**900**). In the horizontal scanning path (**900**), the macroblock pairs (**700**) of a picture (**200**) are coded from left to right and from top to bottom, as shown in FIG. **9**. The other scanning path is a vertical scanning path (**901**). In the vertical scanning path (**901**), the macroblock pairs (**700**) of a picture (**200**) are coded from top to bottom and from left to right, as shown in FIG. **9**. For frame mode coding, the top macroblock of a macroblock pair (**700**) is coded first, followed by the bottom macroblock. For field mode coding, the top field macroblock of a macroblock pair is coded first followed by the bottom field macroblock.

Another embodiment of the present invention extends the concept of AFF coding on a pair of macroblocks to AFF coding on a group of four or more neighboring macroblocks (**902**), as shown in FIG. **10**. AFF coding on a group of macroblocks will be occasionally referred to as group based AFF coding. The same scanning paths, horizontal (**900**) and vertical (**901**), as are used in the scanning of macroblock pairs are used in the scanning of groups of neighboring macroblocks (**902**). Although the example shown in FIG. **10** shows a group of four macroblocks, the group can be more than four macroblocks.

If the group of macroblocks (**902**) is to be encoded in frame mode, the group coded as four frame-based macroblocks. In each macroblock, the two fields in each of the macroblocks are encoded jointly. Once encoded as frames, the macroblocks can be further divided into the smaller blocks of FIGS. **3***a-f* for use in the temporal prediction with motion compensation algorithm.

However, if a group of four macroblocks (**902**), for example, is to be encoded in field mode, it is first split into one top field 32 by 16 pixel block and one bottom field 32 by 16 pixel block. The two fields are then coded separately. The top field block and the bottom field block can now be divided into macroblocks. Each macroblock is further divided into one of the possible block sizes of FIGS. **3***a-f*. Because this process is similar to that of FIG. **8**, a separate figure is not provided to illustrate this embodiment.

In AFF coding at the macroblock level, a frame/field flag bit is preferably included in a picture's bitstream to indicate which mode, frame mode or field mode, is used in the encoding of each macroblock. The bitstream includes information pertinent to each macroblock within a stream, as shown in FIG. **11**. For example, the bitstream can include a picture header (**110**), run information (**111**), and macroblock type (**113**) information. The frame/field flag (**112**) is preferably included before each macroblock in the bitstream if AFF is performed on each individual macroblock. If the AFF is performed on pairs of macroblocks, the frame/field flag (**112**) is preferably included before each pair of macroblock in the bitstream. Finally, if the AFF is performed on a group of macroblocks, the frame/field flag (**112**) is preferably included before each group of macroblocks in the bitstream. One embodiment is that the frame/field flag (**112**) bit is a 0 if frame mode is to be used and a 1 if field coding is to be used. Another embodiment is that the frame/field flag (**112**) bit is a 1 if frame mode is to be used and a 0 if field coding is to be used.

Another embodiment of the present invention entails a method of determining the size of blocks into which the encoder divides a macroblock in macroblock level AFF. A preferable, but not exclusive, method for determining the ideal block size is sum absolute difference (SAD) with or without bias or rate distortion (RD) basis. For example, SAD checks the performance of the possible block sizes and chooses the ideal block size based on its results. The exact method of using SAD with or without bias or RD basis can be easily be performed by someone skilled in the art.

According to an embodiment of the present invention, each frame and field based macroblock in macroblock level AFF can be intra coded or inter coded. In intra coding, the macroblock is encoded without temporally referring to other macroblocks. On the other hand, in inter coding, temporal prediction with motion compensation is used to code the macroblocks.

If inter coding is used, a block with a size of 16 by 16 pixels, 16 by 8 pixels, 8 by 16 pixels, or 8 by 8 pixels can have its own reference pictures. The block can either be a frame or field based macroblock. The MPEG-4 Part 10 AVC/H.264 standard allows multiple reference pictures instead of just two reference pictures. The use of multiple reference pictures improves the performance of the temporal prediction with motion compensation algorithm by allowing the encoder to find a block in the reference picture that most closely matches the block that is to be encoded. By using the block in the reference picture in the coding process that most closely matches the block that is to be encoded, the greatest amount of compression is possible in the encoding of the picture. The reference pictures are stored in frame and field buffers and are assigned reference frame numbers and reference field numbers based on the temporal distance they are away from the current picture that is being encoded. The closer the reference picture is to the current picture that is being stored, the more likely the reference picture will be selected. For field mode coding, the reference pictures for a block can be any top or bottom field of any of the reference pictures in the reference frame or field buffers.

Each block in a frame or field based macroblock can have its own motion vectors. The motion vectors are spatially predictive coded. According to an embodiment of the present invention, in inter coding, prediction motion vectors (PMV) are also calculated for each block. The algebraic difference between a block's PMVs and its associated motion vectors is then calculated and encoded. This generates the compressed bits for motion vectors.

FIG. **12** will be used to explain various preferable methods of calculating the PMV of a block in a macroblock. A current block, E, in FIG. **12** is to be inter coded as well as its neighboring blocks A, B, C, and D. E will refer hereafter to a current block and A, B, C, and D will refer hereafter to E's neighboring blocks, unless otherwise denoted. Block E's PMV is derived from the motion vectors of its neighboring blocks. These neighboring blocks in the example of FIG. **12** are A, B, C, and D. One preferable method of calculating the PMV for block E is to calculate either the median of the motion vectors of blocks A, B, C, and D, the average of these motion vectors, or the weighted average of these motion vectors. Each of the blocks A through E can be in either frame or field mode.

Another preferable method of calculating the PMV for block E is to use a yes/no method. Under the principles of the yes/no method, a block has to be in the same frame or field coding mode as block E in order to have its motion vector included in the calculation of the PMV for E. For example, if block E in FIG. **12** is in frame mode, block A must also be in frame mode to have its motion vector included in the calculation of the PMV for block E. If one of E's neighboring blocks does not have the same coding mode as does block E, its motion vectors are not used in the calculation of block E's PMV.

The "always method" can also be used to calculate the PMV for block E. In the always method, blocks A, B, C, and D are always used in calculating the PMV for block E, regardless of their frame or field coding mode. If E is in frame mode and a neighboring block is in field mode, the vertical component of the neighboring block is multiplied by 2 before being included in the PMV calculation for block E. If E is in field mode and a neighboring block is in frame mode, the vertical component of the neighboring block is divided by 2 before being included in the PMV calculation for block E.

The "selective method" can also be used to calculate the PMV for block E if the macroblock has been encoded using pair based AFF encoding or group based AFF encoding. In the selective method, a frame-based block has a frame-based motion vector pointing to a reference frame. The block is also assigned a field-based motion vector pointing to a reference field. The field-based motion vector is the frame-based motion vector of the block with the vertical motion vector component divided by two. The reference field number is the reference frame number multiplied by two. A field-based block has a field-based motion vector pointing to a reference field. The block is also assigned a frame-based motion vector pointing to a reference frame. The frame-based motion vector is the field-based motion vector of the block with the vertical motion vector component multiplied by two. The reference frame number is the reference field number divided by two.

The derivation of a block's PMV using the selective method will now be explained using FIG. **12** as a reference. In macroblock pair based AFF, each block in a macroblock is associated with a companion block that resides in the same geometric location within the second macroblock of the macroblock pair. In FIG. **12**, each of block E's neighboring blocks (A, B, C, and D) may or may not be in the same frame or field coding mode as block E. Hence, the following rules apply:

If E is in frame mode and a neighboring block is in frame mode, the true frame-based motion vector of the neighboring block is used for E's PMV.

If E is in frame mode and a neighboring block is in field mode, the following rules apply in calculating B's PMV. If the neighboring block (e.g.; black A) and its companion field-based block have the same reference field, the average of the assigned frame-based motion vectors of the two blocks is used for the calculation of E's PMV. The reference frame number used for the PMV calculation is the reference field number of the neighboring block divided by two. However, if the neighboring block and its companion field block have different reference fields, then the neighboring block cannot be used in the calculation of E's PMV.

If E is in field mode and a neighboring block is in frame mode, the following rules apply in calculating E's PMV. If the neighboring block (e.g.; block A) and its companion frame-based block have the same reference frame, the average of the assigned field-based motion vectors of the two blocks is used for the calculation of E's PMV. The reference field number used for the PMV calculation is the reference frame number of the neighboring block multiplied by two. However, if the neighboring block and its compan-

ion field block have different reference frames, then the neighboring block cannot be used in the calculation of E's PMV.

If E is in field mode and a neighboring block is in field mode, the true field-based motion vector of the neighboring block is used in the calculation of E's PMV.

An alternate preferable option can be used in the selective method to calculate a block's PMV. In FIG. **12**, each of block E's neighboring blocks (A, B, C, and D) may or may not be in the same frame or field coding mode as block E. Hence, the following rules apply for this alternate preferable option of the selective method:

If E is in frame mode and a neighboring block is in frame mode, the true frame-based motion vector of the neighboring block is used for E's PMV.

If E is in frame mode and a neighboring block is in field mode, the weighted average of the assigned frame-based motion vectors of the neighboring block and its companion field-based block is used for the calculation of E's PMV. The weighting factors are based upon the reference field numbers of the neighboring block and its companion block.

If E is in field mode, and a neighboring block is in frame mode, the weighted average of the assigned field-based motion vectors of the neighboring block and its companion frame-based block is used for the calculation of E's PMV. The weighting factors are based upon the reference frame numbers of the neighboring block and its companion block.

If E is in field mode and a neighboring block is in field mode, the true field-based motion vector of the neighboring block is used in the calculation of E's PMV.

Another preferable method of computing a block's PMV is the "alt selective method." This method can be used in single macroblock AFF coding, pair based macroblock AFF coding, or group based AFF coding. In this method, each block is assigned a horizontal and a vertical index number, which represents the horizontal and vertical coordinates of the block. Each block is also assigned a horizontal and vertical field coordinate. A block's horizontal field coordinate is same as its horizontal coordinate. For a block in a top field macroblock, the vertical field coordinate is half of vertical coordinate of the block and is assigned top field polarity. For a block in the bottom field macroblock, the vertical field coordinate of the block is obtained by subtracting 4 from the vertical coordinate of the block and dividing the result by 2. The block is also assigned bottom field polarity. The result of assigning different field polarities to two blocks is that there are now two blocks with the same horizontal and vertical field coordinates but with differing field polarities. Thus, given the coordinates of a block, the field coordinates and its field polarity can be computed and vice versa.

The alt selective method will now be explained in detail using FIG. **12** as a reference. The PMV of block E is to be computed. Let bx represent the horizontal size of block E divided by 4, which is the size of a block in this example. The PMVs for E are obtained as follows depending on whether E is in frame/field mode.

Let block E be in frame mode and let (x,y) represent the horizontal and vertical coordinates respectively of E. The neighboring blocks of E are defined in the following manner. A is the block whose coordinates are (x−1,y). B is the block whose coordinates are (x,y−1). D is the block whose coordinates are (x−1,y−1). C is the block whose coordinates are (x+bx+1,y−1). If either A, B, C or D is in field mode then its vertical motion vector is multiplied by 2 before being used for prediction and its reference frame number is computed by dividing its reference field by 2.

Now, let block E be in top or bottom field mode and let (xf,yf) represent the horizontal and vertical field coordinates respectively of E. In this case, the neighbors of E are defined as follows. A is the block whose field coordinates are (xf−1,yf) and has same polarity as E. B is the block whose field coordinates are (xf,yf−1) and has same polarity as E. D is the block whose field coordinates are (xf−1,yf−1) and has same polarity as E. C is the block whose field coordinates are (xf+bx+1,yf) and has same polarity as B. If either A,B,C or D is in frame mode then its vertical motion vector is divided by 2 before being used for prediction and its reference field is computed by multiplying its reference frame by 2.

In all of the above methods for determining the PMV of a block, a horizontal scanning path was assumed. However, the scanning path can also be a vertical scanning path. In this case, the neighboring blocks of the current block, E, are defined as shown in FIG. **13**. A vertical scanning path is preferable in some applications because the information on all the neighboring blocks is available for the calculation of the PMV for the current block E.

Another embodiment of the present invention is directional segmentation prediction. In directional segmentation prediction, 16 by 8 pixel blocks and 8 by 16 pixel blocks have rules that apply to their PMV calculations only. These rules apply in all PMV calculation methods for these block sizes. The rules will now be explained in detail in connection with FIG. **12**. In each of these rules, a current block E is to have its PMV calculated.

First, a 16 by pixel block consists of an upper block and a lower block. The upper block contains the top 8 rows of 16 pixels. The lower block contains the bottom 8 rows of 16 pixels. In the following description, block E of FIG. **12** is a 16 by 8 pixel block. For the upper block having a 16 by 8 pixel block, block B is used to predict block E's PMV if it has the same reference picture as block E. Otherwise, median prediction is used to predict block E's PMV. For the lower block having a 16 by 8 pixel block, block A is used to predict block E's PMV if it has the same reference picture as block E. Otherwise, median prediction is used to predict block E's PMV.

A 16 by 16 pixel block is divided into a right and left block. Both right and left blocks are 8 by 16 pixels. In the following description, block E of FIG. **12** is a 8 by 16 pixel block. For the left block, block A is used to predict block E's PMV if it has the same reference picture as block E. Otherwise, median prediction is used to predict block E's PMV For the right block, block C is used to predict block E's PMV if it has the same referenced picture as block E. Otherwise median prediction is used to predict block E's PMV.

For both 16 by 8 pixel blocks and 8 by 16 blocks, A, B, or C can be in different encoding modes (frame or field) than the current block E. The following rules apply for both block sizes. If E is in frame mode, and A, B, or C is in field mode, the reference frame number of A, B, or C is computed by dividing its reference field by 2. If E is in field mode, and A, B, or C is in frame mode, the reference field number of A, B, or C is computed by multiplying its reference frame by 2.

According to another embodiment of the present invention, a macroblock in a P picture can be skipped in AFF coding. If a macroblock is skipped, its data is not transmitted in the encoding of the picture. A skipped macroblock in a P picture is reconstructed by copying the co-located macroblock in the most recently coded reference picture. The co-located macroblock is defined as the one with motion compensation using PMV as defined above or without

motion vectors. The following rules apply for skipped macroblocks in a P picture. If AFF coding is performed per macroblock, a skipped macroblock is in frame mode. If AFF coding is performed on macroblock pairs and if both macroblocks are skipped, then they are in frame mode. However, if only one of the macroblocks in a macroblock pair is skipped, its frame or field coding mode is the same as the non-skipped macroblock in the same macroblock pair. If AFF coding is performed on a group of macroblocks and if the entire group of macroblocks is skipped, then all the macroblocks are in frame mode. If there is at least one macroblock that is not skipped, then the skipped macroblocks in the same group are in the same frame or field coding mode as the non-skipped macroblock.

An alternate method for skipped macroblocks is as follows. If a macroblock pair is skipped, its frame and field coding mode follows its neighboring macroblock pair to the left. If the left neighboring macroblock pair is not available, its coding mode follows its neighboring macroblock pair to the top. If neither the left nor top neighboring macroblock pairs are available, the skipped macroblock is set to frame mode.

Another embodiment of the present invention is direct mode macroblock coding for B pictures. In direct mode coding, a B picture has two motion vectors, forward and backward motion vectors. Each motion vector points to a reference picture. Both the forward and backward motion vectors can point in the same temporal direction. For direct mode macroblock coding in B pictures, the forward and backward motion vectors of a block are calculated from the co-located block in the backward reference picture. The co-located block in the backward reference picture can be frame mode or field mode coded. The following rules apply in direct mode macroblock coding for B picture.

If the co-located block is in frame mode and if the current direct mode macroblock is also in frame mode, the two associated motion vectors of a block in the direct mode macroblock are calculated from the co-located block. The forward reference frame is the one used by the co-located block. The backward reference frame is the same frame where the co-located block resides.

If the co-located block is in frame mode and if the current direct mode macroblock is in field mode, the two associated motion vectors of a block in the direct mode macroblock are calculated from the co-located block's motion vector with vertical component divided by two. The forward reference field is the same parity field of the reference frame used by the co-located block. The backward reference field is the same parity field of the backward reference frame where the co-located block resides.

If the co-located block is in field mode and if the current direct mode macroblock is also in field mode, the two associated motion vectors of a block in the direct mode macroblock are calculated from the co-located block of the same field parity. The forward reference field is the field used by the co-located block. The backward reference field is the same field where the co-located block resides.

If the co-located block is in field mode and if the current direct mode macroblock is in frame mode, the two associated motion vectors of the block in the direct mode macroblock are calculated from the co-located block's motion vector with vertical component multiplied by two. The forward reference frame is the frame one of whose fields is used by the co-located block. The backward reference field is the frame in one of whose fields the co-located block resides.

An alternate option is to force the direct mode block to be in the same frame or field coding mode as the co-located block. In this case, if the co-located block for a direct mode block is in frame mode, the direct mode block is in frame mode as well. The two frame-based motion vectors of the direct mode block are derived from the frame-based forward motion vector of the co-located block. The forward reference frame is used by the co-located block. The backward reference frame is where the co-located block resides.

However, if the co-located block for a block in direct mode is in field mode, the direct mode block is also in field mode. The two field-based motion vectors of the direct mode block are derived from the field-based forward motion vector of the co-located block. The forward reference field is used by the co-located block. The backward reference field is where the co-located block resides.

A macroblock in a B picture can also be skipped in AFF coding according to another embodiment of the present invention. A skipped macroblock in a B picture is reconstructed as a regular direct mode macroblock without any coded transform coefficient information. For skipped macroblocks in a B picture, the following rules apply. If AFF coding is performed per macroblock, a skipped macroblock is either in frame mode or in the frame or field coding mode of the co-located block in its backward reference picture. If AFF coding is performed on macroblock pairs and if both macroblocks are skipped, then they are in frame mode or in the frame or field coding mode of the co-located macroblock pair in the its backward reference picture. However, if only one of the macroblocks in a macroblock pair is skipped, its frame or field coding mode is the same as the non-skipped macroblock of the same macroblock pair. If AFF coding is performed on a group of macroblocks and if the entire group of macroblocks is skipped, then all the macroblocks are in frame mode or in the frame or field coding mode of the co-located group of macroblocks in the backward reference picture. If there is at least one macroblock that is not skipped, then the skipped macroblock in the same group are in the same frame or field coding mode as the non-skipped macroblock.

As previously mentioned, a block can be intra coded. Intra blocks are spatially predictive coded. There are two possible intra coding modes for a macroblock in macroblock level AFF coding. The first is intra_4×4 mode and the second is intra_16×16 mode. In both, each pixel's value is predicted using the real reconstructed pixel values from neighboring blocks. By predicting pixel values, more compression can be achieved. The intra_4×4 mode and the intra_16×16 modes will each be explained in more detail below.

For intra_4×4 mode, the predictions of the pixels in a 4 by 4 pixel block, as shown in FIG. **14**, are derived form its left and above pixels. In FIG. **14**, the 16 pixels in the 4 by 4 pixel block are labeled a through p. Also shown in FIG. **14** are the neighboring pixels A through P. The neighboring pixels are in capital letters. As shown in FIG. **15**, there are nine different prediction directions for intra_4×4 coding. They are vertical (**0**), horizontal (**1**), DC prediction (mode 2), diagonal down/left (**3**), diagonal down/right (**4**), vertical-left (**5**), horizontal-down (**6**), vertical-right (**7**), and horizontal-up (**8**). DC prediction averages all the neighboring pixels together to predict a particular pixel value.

However, for intra_16×16 mode, there are four different prediction directions. Prediction directions are also referred to as prediction modes. These prediction directions are vertical prediction (**0**), horizontal prediction (**1**), DC prediction, and plane prediction. Plane prediction will not be explained.

An intra block and its neighboring blocks may be coded in frame or field mode. Intra prediction is performed on the reconstructed blocks. A reconstructed block can be represented in both frame and field mode, regardless of the actual frame or field coding mode of the block. Since only the pixels of the reconstructed blocks are used for intra prediction, the following rules apply.

If a block of 4 by 4 pixels or 16 by 16 pixels is in frame mode, the neighboring pixels used in calculating the pixel value predictions of the block are in the frame structure. If a block of 4 by 4 pixels or 16 by 16 pixels is in field mode, the neighboring pixels used in calculating the pixel value prediction of the block are in field structure of the same field parity.

The chosen intra-prediction mode (intra_pred_mode) of a 4 by 4 pixel block is highly correlated with the prediction modes of adjacent blocks. This is illustrated in FIG. **16***a*. FIG. **16***a* shows that A and B are adjacent blocks to C. Block C's prediction mode is to be established. FIG. **16***b* shows the order of intra prediction information in the bitstream. When the prediction modes of A and B are known (including the case that A or B or both are outside the slice) the most probable prediction mode (most_probable_mode) of C is given. If one of the blocks A or B is "outside" the most_probable prediction mode is equal DC prediction (mode 2). Otherwise it is equal to the minimum of prediction modes used for blocks A and B. When an adjacent block is coded by 16×16 intra mode, prediction mode is DC prediction mode. When an adjacent block is coded a non-intra macroblock, prediction mode is "mode 2: DC prediction" in the usual case and "outside" in the case of constrained intra update.

To signal a prediction mode number for a 4 by 4 block first parameter use_most_probable_mode is transmitted. This parameter is represented by 1 bit codeword and can take values 0 or 1. If use_most_probable_mode is equal to 1 the most probable mode is used. Otherwise an additional parameter remaining_mode_selector, which can take value from 0 to 7 is sent as 3 bit codeword. The codeword is a binary representation of remaining_mode_selector value. The prediction mode number is calculated as:

```
if (remaining_mode_selector<most_probable mode)
intra_pred_mode=remaining_mode_selector;
else
intra_pred_mode=remaining_mode_selector+1;
```

The ordering of prediction modes assigned to blocks C is therefore the most probable mode followed by the remaining modes in the ascending order.

An embodiment of the present invention includes the following rules that apply to intra mode prediction for an intra-prediction mode of a 4 by 4 pixel block or an intra-prediction mode of a 16 by 16 pixel block. Block C and its neighboring blocks A and B can be in frame or field mode. One of the following rules shall apply. FIGS. **16***a-b* will be used in the following explanations of the rules.

Rule 1: A or B is used as the neighboring block of C only if A or B is in the same frame/field mode as C. Otherwise, A or B is considered as outside.

Rule 2: A and B are used as the neighboring blocks of C, regardless of their frame/field coding mode.

Rule 3: If C is coded in frame mode and has co-ordinates (x,y), then A is the block with co-ordinates (x,y-1) and B is the block with co-ordinates (x-1,y). Otherwise, if C is coded as field and has field co-ordinates (xf,yf) then A is the block whose field co-ordinates are (xf,yf-1) and has same field polarity as C and B is the block whose field co-ordinates are (xf-1,yf) and has same field polarity as C.

Rule 4: This rule applies to macroblock pairs only. In the case of decoding the prediction modes of blocks numbered 3, 6, 7, 9, 12, 13, 11, 14 and 15 of FIG. **16***b*, the above and the left neighboring blocks are in the same macroblock as the current block. However, in the case of decoding the prediction modes of blocks numbered 1, 4, and 5, the top block (block A) is in a different macroblock pair than the current macroblock pair. In the case of decoding the prediction mode of blocks numbered 2, 8, and 10, the left block (block B) is in a different macroblock pair. In the case of decoding the prediction mode of the block numbered 0, both the left and the above blocks are in different macroblock pairs. For a macroblock in field decoding mode the neighboring blocks of the blocks numbered 0, 1, 4, 5, 2, 8, and 10 shall be defined as follows:

If the above macroblock pair (**170**) is decoded in field mode, then for blocks number 0, 1, 4 and 5 in the top-field macroblock (**173**), blocks numbered 10, 11, 14 and 15 respectively in the top-field macroblock (**173**) of the above macroblock pair (**170**) shall be considered as the above neighboring blocks to the current macroblock pair (**171**) as shown in FIG. **17***a*. For blocks number 0, 1, 4 and 5 in the bottom-field macroblock (**174**), blocks numbered 10, 11, 14 and 15 respectively in the bottom-field MB of the above macroblock pair (**170**) shall be considered as the above neighboring blocks to the current macroblock pair (**171**), as shown in FIG. **17***a*.

However, if the above macroblock pair (**170**) is decoded in frame mode then for blocks number 0, 1, 4 and 5 in the top-field macroblock (**173**), blocks numbered 10, 11, 14 and 15 respectively in the bottom-frame macroblock (**176**) of the above macroblock pair (**170**) shall be considered as the above neighboring blocks to the current macroblock pair (**171**), as shown in FIG. **17***b*. For blocks number 0, 1, 4 and 5 in the bottom-field macroblock (**174**), blocks numbered 10, 11, 14 and 15 respectively in the bottom-frame macroblock (**176**) of the above macroblock pair (**170**) shall be considered as the above neighboring blocks to the current macroblock pair (**171**), as shown inn FIG. **17***b*.

If the left macroblock pair (**172**) is decoded in field mode, then for blocks number 0, 2, 8 and 10 in the top-field macroblock (**173**), blocks numbered 5, 7, 13 and 15 respectively in the top-field macroblock (**173**) of the left macroblock pair (**172**) shall be considered as the left neighboring blocks to the current macroblock pair (**171**) as shown in FIG. **17***c*. For blocks number 0, 2, 8 and 10 in the bottom-field macroblock (**174**), blocks numbered 5, 7, 13 and 15 respectively in the bottom-field macroblock (**174**) of the left macroblock pair (**172**) shall be considered as the left neighboring blocks to the current macroblock pair (**171**), as shown in FIG. **17***c*.

If the left macroblock pair (**172**) is decoded in frame mode, then for blocks number 0, 2, 8 and 10 in the top-field macroblock (**173**), the blocks numbered 5, 7, 13 and 15 respectively in the top-frame macroblock (**175**) of the left macroblock pair (**172**) shall be considered as the left neighboring blocks to the current macroblock pair (**171**), as shown in FIG. **17***d*. For blocks number 0, 2, 8 and 10 in the bottom-field macroblock (**174**), blocks numbered 5, 7, 13 and 15 respectively in the bottom-frame macroblock (**176**) of the left macroblock pair (**172**) shall be considered as the left neighboring blocks to the current macroblock pair (**171**), as shown in FIG. **17***d*.

For macroblock pairs on the upper boundary of a slice, if the left macroblock pair (**172**) is in frame decoding mode, then the intra mode prediction value used to predict a field macroblock shall be set to DC prediction.

The preceding descriptions of intra coding and intra mode prediction can be extended to adaptive block transforms.

Another embodiment of the present invention is that loop filtering is performed on the reconstructed blocks. A reconstructed block can be represented in either frame or field structure, regardless of the frame/field coding mode of the block. Loop (deblock) filtering is a process of weighted averaging of the pixels of the neighboring blocks. FIG. **12** will be used to explain loop filtering. Assume E of FIG. **12** is a reconstructed block, and A, B, C and D are its neighboring reconstructed blocks, as shown in FIG. **12**, and they are all represented in frame structure. Since A, B, C, D and B can be either frame- or field-coded, the following rules apply:

Rule 1: If E is frame-coded, loop filtering is performed over the pixels of E and its neighboring blocks A B, C and D.

Rule 2: If E is field-coded, loop filtering is performed over the top-field and bottom-field pixels of E and its neighboring blocks A B, C and D, separately.

Another embodiment of the present invention is that padding is performed on the reconstructed frame by repeating the boundary pixels. Since the boundary blocks may be coded in frame or field mode, the following rules apply:

Rule 1: The pixels on the left or right vertical line of a boundary block are repeated, if necessary.

Rule 2: If a boundary block is in frame coding, the pixels on the top or bottom horizontal line of the boundary block are repeated.

Rule 3: if a boundary block is in field coding, the pixels on the two top or two bottom horizontal (two field) lines of the boundary block are repeated alternatively.

Another embodiment of the present invention is that two-dimensional transform coefficients are converted into one-dimensional series of coefficients before entropy coding. The scan path can be either zigzag or non-zigzag. The zigzag scanner is preferably for progressive sequences, but it may be also used for interlace sequences with slow motions. The non-zigzag scanners are preferably for interlace sequences. For macroblock AFF coding, the following options may be used:

Option 1: The zigzag scan is used for macroblocks in frame mode while the non-zigzag scanners are used for macroblocks in field coding.

Option 2: The zigzag scan is used for macroblocks in both frame and field modes.

Option 3: The non-zigzag scan is used for macroblocks in both frame and field modes.

The preceding description has been presented only to illustrate and describe embodiments of invention. It is not intended to be exhaustive or to limit the invention to any precise form disclosed. Many modifications and variations are possible in light of the above teaching.

The foregoing embodiments were chosen and described in order to illustrate principles of the invention and some practical applications. The preceding description enables others skilled in the art to utilize the invention in various embodiments and with various modifications as are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the following claims.

What is claimed is:

1. A method of encoding a picture in an image sequence, comprising:

dividing said picture into a plurality of macroblocks, each macroblock containing a plurality of blocks;

generating a plurality of processing blocks, each processing block being generated by grouping said plurality of macroblocks as a processing block, said plurality of macroblocks including a pair of macroblocks or a group of macroblocks; and

selectively encoding at least one of said processing blocks at a time in frame coding mode and at least one of said processing blocks at a time in field coding mode,

wherein said encoding is performed in a horizontal scanning path or a vertical scanning path.

2. The method of claim **1**, wherein said processing block includes a pair of macroblocks.

3. The method of claim **2**, wherein each pair of macroblocks of said image is encoded from left to right and from top to bottom if said encoding is performed in said horizontal scanning path, and

wherein each pair of macroblocks of said image is encoded from top to bottom and from left to right if said encoding is performed in said vertical scanning path.

4. The method of claim **2**, wherein said pair of macroblocks comprises a top block and a bottom block, where said top block is encoded prior to said bottom block in said frame coding mode.

5. The method of claim **2**, further comprising:

splitting said pair of macroblocks into a top field block and a bottom field block when said pair of macroblocks are encoded in said field coding mode, and where said top field block is encoded prior to said bottom field block.

6. The method of claim **1**, wherein said processing block includes a group of at least four macroblocks blocks.

7. An apparatus for encoding a picture in an image sequence, comprising:

means for dividing said picture into a plurality of macroblocks, each macroblock containing a plurality of blocks; and

means for generating a plurality of processing blocks, each processing block being generated by grouping said plurality of macroblocks as a processing block, said plurality of macroblocks including a pair of macroblocks or a group of macroblocks;

means for selectively encoding at least one of said processing blocks at a time in frame coding mode and at least one of said processing blocks at a time in field coding mode,

wherein said encoding is performed in a horizontal scanning path or a vertical scanning path.

8. The apparatus of claim **7** wherein said processing block includes a pair of macroblocks.

9. The apparatus of claim **8**, wherein each pair of macroblocks of said image is encoded from left to right and from top to bottom if said encoding is performed in said horizontal scanning path, and

wherein each pair of macroblocks of said image is encoded from top to bottom and from left to right if said encoding is performed in said vertical scanning path.

10. The apparatus of claim **8**, wherein said pair of macroblocks comprises a top block and a bottom block, where said top block is encoded prior to said bottom block in said frame coding mode.

11. The appartus of claim **8**, wherein said pair of macroblocks are split into a top field block and a bottom field block when said pair of macroblocks are encoded in said field coding mode, and where said top field block is encoded prior to said bottom field block.

12. The apparatus of claim **7**, wherein said processing block includes a group of at least four macroblocks blocks.

13. A computer-readable medium encoded with computer executable instructions, the computer executable instruc-

tions including instructions which, when executed by a processor, cause the processor to perform the steps of a method for encoding a picture in an image sequence, comprising the steps of:

dividing said picture into a plurality of macroblocks, each macroblock containing a plurality of blocks;

generating a plurality of processing blocks, each processing block being generated by grouping said plurality of macroblocks as a processing block, said plurality of macroblocks including a pair of macroblocks or a group of macroblocks; and

selectively encoding at least one of said processing blocks at a time in frame coding mode and at least one of said processing blocks at a time in field coding mode,

wherein said encoding is performed in a horizontal scanning path or a vertical scanning path.

14. A method of decoding an encoded picture having a plurality of processing blocks, each processing block containing macroblocks, each macroblock containing a plurality of blocks, from a bitstream, comprising:

decoding at least one of a plurality of processing blocks at a time, wherein each of said plurality of processing blocks includes a pair of macroblocks or a group of macroblocks, in frame coding mode and at least one of said plurality of processing blocks at a time in field coding mode, wherein said decoding is applied to a pair of blocks, or a group of blocks,

wherein said decoding is performed in a horizontal scanning path or a vertical scanning path; and

using said plurality of decoded processing blocks to construct a decoded picture.

15. The method of claim **14**, wherein said processing blocks include a pair of macroblocks.

16. The method of claim **14**, wherein said decoding is applied to a group of at least four macroblocks.

17. The method of claim **16**, further comprising: joining said group of at least four macroblocks into a top field block and a bottom field block when said group of at least four macroblocks is decoded in said field coding mode.

18. The method of claim **15**, wherein each pair of macroblocks of said image is decoded from left to right and from top to bottom if said decoding is performed in said horizontal scanning path, and

wherein each pair of macroblocks of said image is decoded from top to bottom and from left to right if said decoding is performed in said vertical scanning path.

19. The method of claim **15**, wherein said pair of macroblocks comprises a top block and a bottom block, where said top block is decoded prior to said bottom block in said frame coding mode.

20. The method of claim **15**, wherein said pair of macroblocks is represented by a top field block and a bottom field block in said field coding mode, the method further comprising:

decoding said top field block and said bottom field block, and

joining said top field block and said bottom field block into said pair of macroblocks.

21. The method of claim **14**, wherein said processing block includes a group of at least four macroblocks blocks.

22. An apparatus for decoding an encoded picture from a bitstream, comprising:

means for decoding at least one of a plurality of processing blocks at a time, each processing block containing a pair of macroblocks or a group of macroblocks, each macroblock containing a plurality of blocks, from said encoded picture that is encoded in frame coding mode and at least one of said plurality of processing blocks at a time that is encoded in field coding mode,

wherein said decoding is performed in a horizontal scanning path or a vertical scanning path; and

means for using said plurality of decoded processing blocks to construct a decoded picture.

23. The apparatus of claim **22**, wherein said processing blocks include a pair of macroblocks.

24. The apparatus of claim **22**, wherein said decoding is applied to a group of at least four macroblocks.

25. The apparatus of claim **24**, further comprising: means for joining said group of at least four macroblocks into a top field block and a bottom field block when said group of at least four macroblocks is decoded in said field coding mode.

26. The apparatus of claim **23**, wherein each pair of macroblocks of said image is decoded from left to right and from top to bottom if said decoding is performed in said horizontal scanning path, and

wherein each pair of macroblocks of said image is decoded from top to bottom and from left to right if said decoding is performed in said vertical scanning path.

27. The apparatus of claim **23**, wherein said pair of macroblocks comprises a top block and a bottom block, where said top block is decoded prior to said bottom block in said frame coding mode.

28. The apparatus of claim **23**, wherein said pair of macroblocks is represented by a top field block and a bottom field block in said field coding mode, the method further comprising:

decoding said top field block and said bottom field block, and

joining said top field block and said bottom field block into said pair of macroblocks.

29. The apparatus of claim **22**, wherein said processing block includes a group of at least four macroblocks blocks.

30. A computer-readable medium encoded with computer executable instructions, the computer executable instructions including instructions which, when executed by a processor, cause the processor to perform the steps of a method for encoding a picture in an image sequence, comprising the steps of:

decoding at least one of a plurality of processing blocks at a time, each processing block containing a pair of macroblocks or a group of macroblocks, each macroblock containing a plurality of blocks, from said encoded picture that is encoded in frame coding mode and at least one of said plurality of processing blocks at a time that is encoded in field coding mode,

wherein said decoding is performed in a horizontal scanning path or a vertical scanning path; and

using said plurality of decoded processing blocks to construct a decoded picture.

31. A bitstream comprising:

a picture that has been divided into a plurality of processing blocks, each processing block containing a pair of macroblocks or a group of macroblocks, each macroblock containing a plurality of blocks,

wherein at least one of said plurality of processing blocks from said picture is encoded in frame coding mode at a time and at least one of said plurality of processing blocks is encoded in field coding mode at a time,

wherein said encoding is performed in a horizontal scanning path or a vertical scanning path.

* * * * *

Exhibit D

To

Joint Claim Chart

US006339780B1

(12) **United States Patent**
Shell et al.

(10) **Patent No.: US 6,339,780 B1**
(45) **Date of Patent: Jan. 15, 2002**

(54) **LOADING STATUS IN A HYPERMEDIA BROWSER HAVING A LIMITED AVAILABLE DISPLAY AREA**

(75) Inventors: **Scott R. Shell; Kevin Timothy Shields**, both of Redmond; **Anthony Kitowitz**, Kirkland, all of WA (US)

(73) Assignee: **Microsoft Corporation**, Redmond, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **08/851,877**

(22) Filed: **May 6, 1997**

(51) **Int. Cl.**[7] **G06F 17/00**
(52) **U.S. Cl.** **707/526**; 707/102
(58) **Field of Search** 707/1–576; 345/24–440

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,266,253	A *	5/1981	Matheral	345/24
5,467,459	A *	11/1995	Alexander et al.	345/514
5,731,813	A *	3/1998	O'Rourke et al.	345/349
5,760,771	A *	6/1998	Blonder et al.	345/302
5,774,666	A *	6/1998	Portuesi	709/218
5,877,766	A *	3/1999	Bates et al.	345/357
5,973,692	A *	10/1999	Knowlton et al.	345/348
5,983,005	A *	11/1999	Monteiro et al.	709/231
6,101,510	A *	8/2000	Stone et al.	707/513

OTHER PUBLICATIONS

Smallman et al. "Information availability in 2D and 3D displays", IEEE Computer Graphics and Applications, vol. 21 Issue 5, Sep./Oct. 2001, pp. 51–57.*
Hu et al., "Parameterizable fonts based on shape components", IEEE Computer Graphics and Applications, vol. 21 Issue 3, May/Jun. 2001, pp. 70–85.*
Liu et al., "Web-based peer review: the learner as both adapter and reviewer", Education, IEEE Transactions on, vol. 44 Issue 3, Aug. 2001, pp. 246–251.*
www.sciam.com/200/1100issue/1100stjohnbox1.html.*

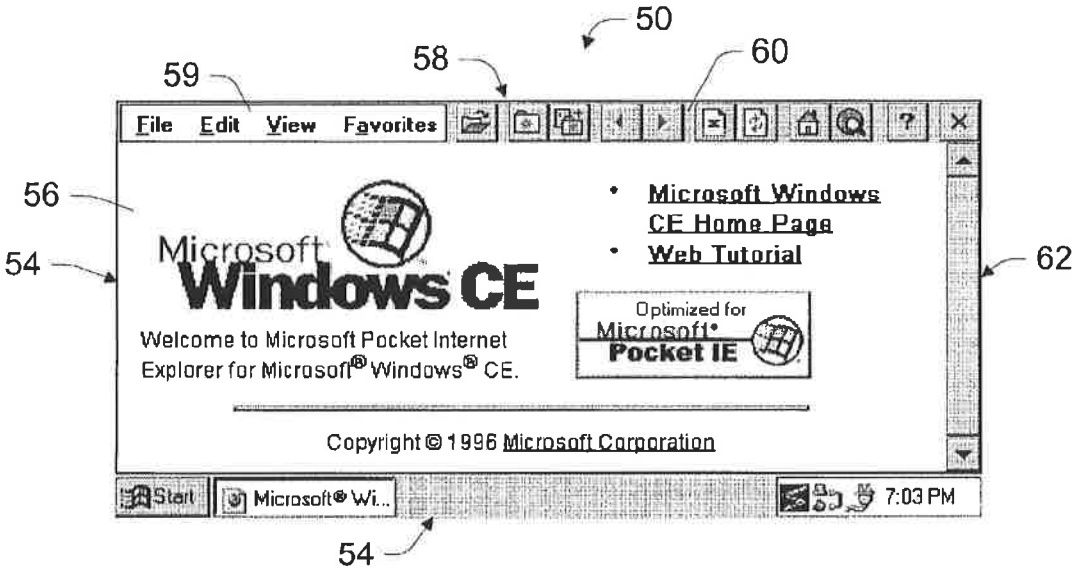
* cited by examiner

Primary Examiner—Thomas Black
Assistant Examiner—David Jung
(74) *Attorney, Agent, or Firm*—Lee & Hayes, PLLC

(57) **ABSTRACT**

Described herein is a portable computer having a limited display area. An Internet or other hypermedia browser executes on the portable computer to load and display content in a content viewing area. During times when the browser is loading content, the browser displays a temporary, animated graphic element over the content viewing area. The graphic element is removed after the content is loaded, allowing unobstructed viewing of the loaded content.

42 Claims, 3 Drawing Sheets



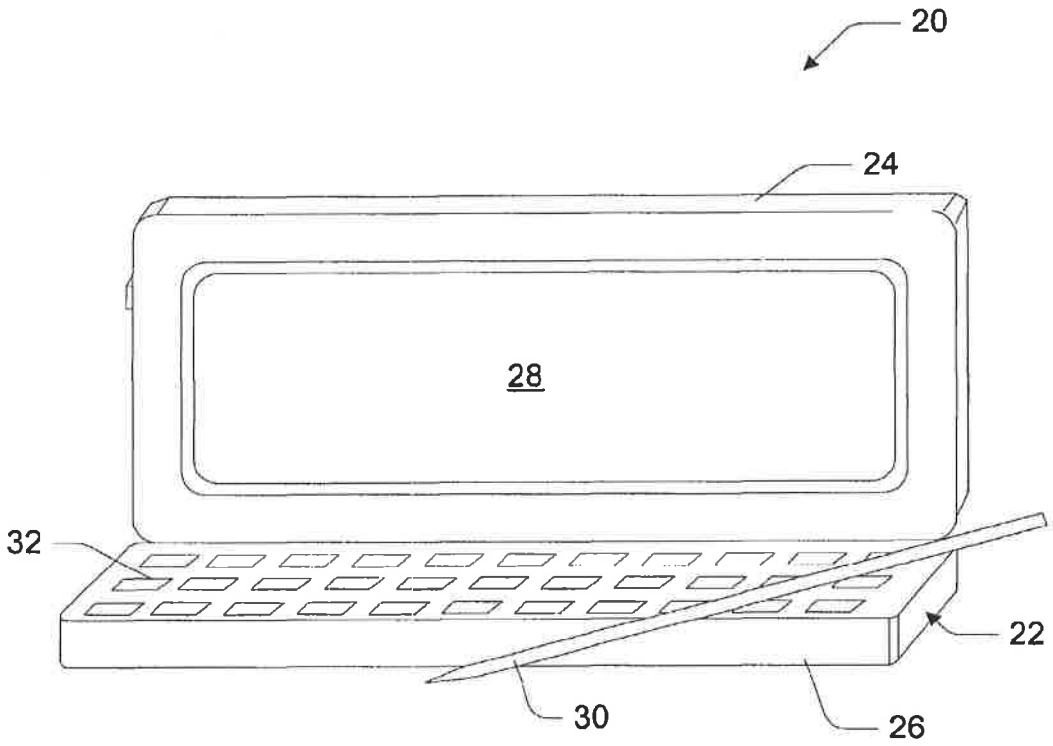


Fig. 1

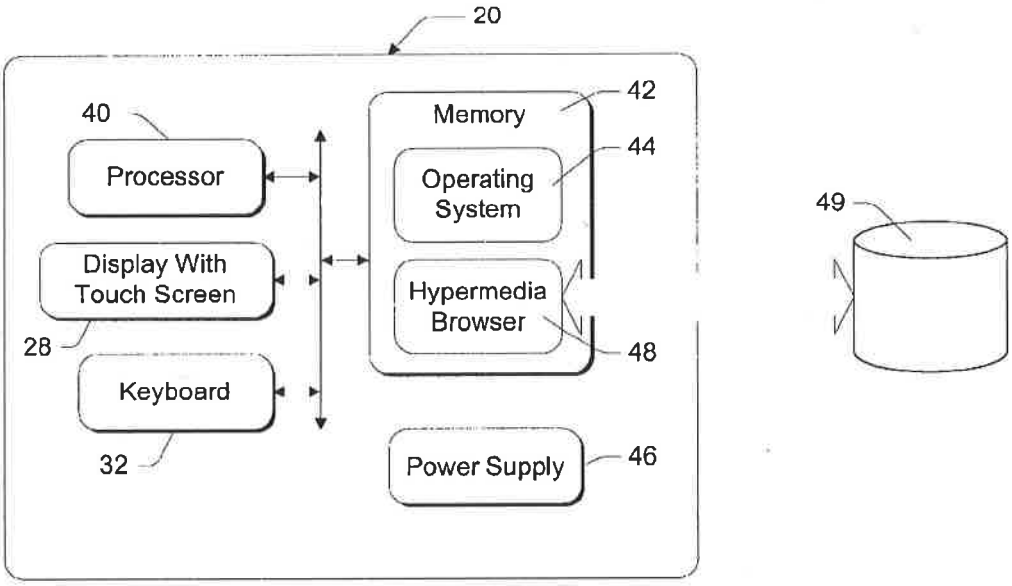

20
40
Processor
Display With Touch Screen
28
Keyboard
32
Memory
42
Operating System
44
Hypermedia Browser
48
Power Supply
46
49

Fig. 2

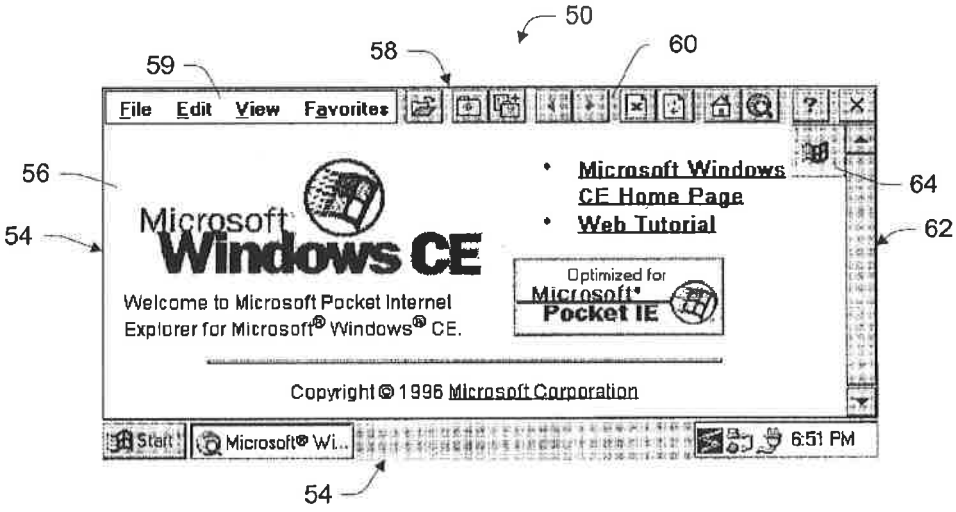

50
58
59
60
File Edit View Favorites
56
54
Microsoft Windows CE
Welcome to Microsoft Pocket Internet Explorer for Microsoft® Windows® CE.
Microsoft Windows CE Home Page
Web Tutorial
64
62
Optimized for Microsoft® Pocket IE
Copyright © 1996 Microsoft Corporation
Start
Microsoft® Wi...
6:51 PM
54

Fig. 3

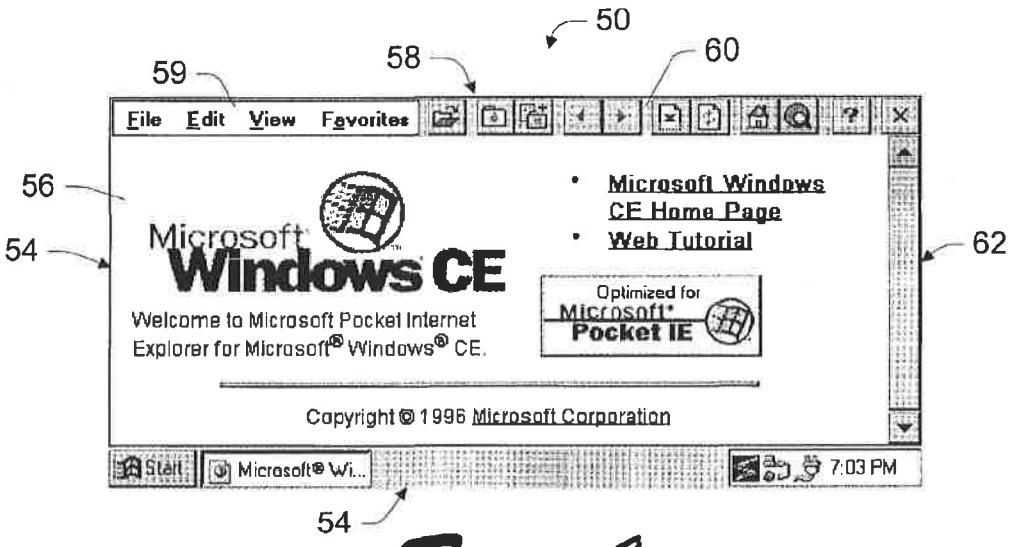

50
58
59
60
File Edit View Favorites
56
54
Microsoft Windows CE
Welcome to Microsoft Pocket Internet Explorer for Microsoft® Windows® CE.
Microsoft Windows CE Home Page
Web Tutorial
Optimized for Microsoft Pocket IE
62
Copyright © 1996 Microsoft Corporation
Start
Microsoft® Wi...
7:03 PM
54

Fig. 4

LOADING STATUS IN A HYPERMEDIA BROWSER HAVING A LIMITED AVAILABLE DISPLAY AREA

TECHNICAL FIELD

This invention relates hypermedia content browsers such as World Wide Web browsers.

BACKGROUND OF THE INVENTION

"Hypermedia" is a metaphor for presenting information in which text, images, sounds, and actions become linked together in a complex, non-sequential web of associations that permit a user to browse through related content and topics, regardless of the presented order of the topics. The term "hypermedia" arises from the similar term "hypertext," which was originally coined to describe the linked text-based documents.

Hypermedia content is widely used for navigation and information dissemination on the "World-Wide Web" (WWW or Web) of the Internet. An application program referred to as a hypermedia browser, hypertext browser, "Web browser" is normally used to retrieve and render hypermedia content from the WWW, although such a browser is also useful for browsing hyperlinked content from other sources.

Hypermedia content is commonly organized as documents with embedded control information. The embedded control information includes formatting specifications, indicating how a document is to be rendered by the Web browser. In addition, such control information can include links or "hyperlinks": symbols or instructions telling the Web browser where to find other related WWW documents. A hyperlink from one hypermedia topic to another is normally established by the author of a hypermedia document, although some applications allow users to insert hyperlinks to desired topics.

A hyperlink is typically rendered by a Web browser as a graphical icon or as highlighted keywords. A user "activates" or "follows" a hyperlink by clicking on or otherwise selecting the icon or highlighted keywords. Activating a link causes the Web browser to load and render the document or resource that is targeted by the hyperlink.

Hyperlink usage is not limited to the Internet. Various multimedia applications and other hypermedia resources utilize hypertext to allow users to navigate through different pieces of information content. For instance, an encyclopedia program might use hyperlinks to provide cross-references to related articles within an electronic encyclopedia. The same program might also use hyperlinks to specify remote information resources such as WWW documents.

Hypermedia browsers have evolved in recent years and are available from several sources. Microsoft's Internet Explorer is one example of a popular browser that is particularly suitable for browsing the WWW and other similar network resources. Browsers such as the Internet Explorer typically have a content viewing area or window, in which textual or other graphical content is displayed. Browser controls such as menus, status displays, and tool icons are located in areas or windows adjacent the viewing area, so that they do not obstruct or interfere with the viewing area.

One persistent characteristic of WWW browsing is that significant delays are often encountered when loading documents and other multimedia content. From the user's perspective, such delays can be quite frustrating. In severe cases involving long delays, users might be inclined to believe that their browsers have become inoperative. To avoid this situation, browsers typically include some type of status display indicating progress in loading content. In many browsers, this consists of a stationary icon such as a flag or globe that becomes animated during periods when content is being loaded. For instance, such an icon might comprise a flag that is normally stationary but that flutters or waves during content loading. An icon such as this is positioned in a tool area or status area outside of the content viewing area. The icon is visible at all times, but is animated only when content is being loaded.

One very recent development relating to this subject is the emergence of a number of popular, small, handheld computing devices that potentially support Internet browsing. These include palmtops, pocket computers, personal digital assistants, personal organizers, and the like. In this disclosure, this class of computing devices is generally referred to as "handheld personal computers", "handheld PCs", or "H/PCs".

One of the most desirable characteristics of H/PCs is their portability. The compact, portable H/PCs provide a user with real computer-like applications—such as email, PIM (personal information management), spreadsheet, and word processing. Hypermedia browsers are among the application programs available for H/PCs. A traveling user can receive email messages, schedule meetings or appointments, and browse the Internet from the H/PC.

To keep H/PCs small, compromises are of course necessary. Chief among the design compromises is an undersized display. Screen space is very limited. Traditional user interface techniques which users are accustomed to on desktop computers are not available for H/PC displays due to the limited size. Additionally, the screen must be efficiently utilized to enable effective data input from the stylus.

With a hypermedia or Internet browser, in particular, there may not be room enough on the available display to implement an animated status display such as described above.

The inventors, however, have developed a method of implementing a status display even within the limited display areas available on popular H/PCs.

SUMMARY OF THE INVENTION

In accordance with the invention, a browser has a content viewing area that is used for displaying graphical hypermedia content. A temporary, animated graphic element is presented in a corner of the content viewing area during times when the browser is loading content. The graphic element is not displayed during any other times.

BRIEF DESCRIPTION OF THE DRAWINGS

The same reference numbers are used throughout the drawings to reference like components and features.

FIG. **1** is a perspective view of a handheld computing device in an open position.

FIG. **2** is a block diagram of the handheld computing device.

FIGS. **3** and **4** are illustrations of displays generated by a hypermedia browser in accordance with the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

FIG. **1** shows a handheld computing device **20**. As used herein, "handheld computing device" means a small com-

puting device having a processing unit that is capable of running one or more application programs, a display, and an input mechanism such as a keypad, a touch-sensitive screen, a track ball, a touch-sensitive pad, a miniaturized QWERTY keyboard, or the like.

The handheld computing device **20** is embodied as a handheld personal computer. The terms "handheld computing device" and "handheld personal computer" (or handheld PC or H/PC) are used interchangeably throughout this disclosure. However, in other implementations, the handheld computing device may be implemented as a personal digital assistant (PDA), a personal organizer, a palmtop computer, a computerized notepad, or the like. The invention can also be implemented in other types of computers and computer-like or computer-controlled devices having graphical display surfaces.

Handheld computing device **20** has a casing **22** with a cover or lid **24** and a base **26**. A liquid crystal display (LCD) **28** with a touch-sensitive screen is mounted to lid **24**. Lid **24** is hinged to base **26** to pivot between an open position, which exposes display **28**, and a closed position, which protects the display. The device is equipped with a stylus **30** to enter data through touchscreen display **28** and a miniature QWERTY keyboard **32**. Stylus **30** and keyboard **32** are both mounted in base **26**. Although the illustrated implementation shows a two-member H/PC **20** with a lid **24** and a base **26**, other implementations of the H/PC might comprise an integrated body without hinged components, as is the case with computerized notepads (e.g., Newton® from Apple Computers).

FIG. **2** shows functional components of the handheld computing device. It has a processor **40**, a computer-readable storage medium or memory **42**, a display **28**, and a keyboard **32**. Memory **42** generally includes both volatile memory (e.g., RAM) and non-volatile memory (e.g., ROM, PCMCIA cards, etc.). The H/PC **20** has a power supply **46** that supplies power to the electronic components. The power supply **46** is preferably implemented as one or more batteries. The power supply **46** might further represent an external power source that overrides or recharges the built-in batteries, such as an AC adapter or a powered docking cradle.

An operating system program **44** is resident in the memory **42** and executes on the processor **40**. The operating system **44** is a multitasking operating system that allows simultaneous execution of multiple applications. The operating system employs a graphical user interface windowing environment that presents applications and documents in specially delineated areas of the display screen called "windows." Each window can act independently, including its own menu, toolbar, pointers, and other controls, as if it were a virtual display device. The handheld computing device may be implemented with other types of operating systems that support a graphical user interface.

The operating system **44** is preferably the Windows® CE operating system from Microsoft Corporation that is configured to execute application programs such as application program **48** shown in FIG. **2**. The Windows® CE operating system is a derivative of Windows® brand operating systems, such as Windows® 95, that is especially designed for handheld computing devices having limited display areas.

In the described embodiment of the invention, application program **48** is an Internet or other hypermedia browser. The browser is stored as a sequence of program instructions in memory **42**, for execution by processor **40**. In other embodiments, the browser might be stored on a portable or removable type of computer-readable storage medium such as a floppy disk or EPROM (erasable read-only memory). As used here, the term "hypermedia browser" refers to an application or application program that is capable of displaying or otherwise rendering hypermedia content and of loading additional or alternative hypermedia content in response to a user's selection of hyperlinks.

Browser **48** has access to a hypermedia resource **49**. Often, this resource will be the Internet. However, other sources of hyperlinked content are frequently available and can be efficiently browsed in accordance with the invention. Computer **20** includes a network interface or modem (not shown) for accessing the hypermedia resource.

FIG. **3** shows an example of a graphical display **50** generated by a hypermedia browser **48** in conjunction with operating system **44**. The display includes a number of elements that are generated by making appropriate system calls to the operating system in accordance with well-known protocols. Specifically, Windows® CE supports a subset of the Win32 API set used in the Windows® 95 operating system. These APIs allow an application program to create a variety of on-screen controls with minimal effort.

In this case, the graphical display **50** includes a taskbar **52** presented by the Windows® CE operating system. Browser **48** presents a main window **54**, above taskbar **52**. Browser main window **54** in this example has three primary components. The largest screen area is dedicated to a content viewing area **56**. This is the area in which graphical hypermedia content is displayed.

Content viewing area **56** is bordered along its upper edge by a toolbar **58**. Toolbar **58** is similar in appearance to toolbars used in other application programs designed for the Windows® operating environment, with some characteristics that are unique to the Windows® CE environment. One characteristic that is unique to Windows® CE is that the toolbar includes both a menu area **59** and an icon area **60**. In previous versions of Windows®, these features were presented within their own distinct areas. Another Windows® CE characteristic is that the toolbar is located on what would have been the "title bar" of previous Windows® application programs. The toolbar thus includes an "X" icon **61** that is used to close the browser application. In previous versions of Windows®, the toolbar would have been below or otherwise separate from the title bar.

A scroll bar **62** borders content viewing area **56** along its right side. Scroll bar **62** is used to vertically scroll the content that is presented in content viewing area **56**.

In contrast to prior art hypermedia browsers, browser **48** does not include a permanent "loading status" icon. In fact, no portion of main window **54** is dedicated permanently to displaying loading status. Rather, the browser is configured to display a temporary graphic element **64** over content viewing area **56** during times when the browser is loading content. This temporary graphic element is preferably animated (such as the waving Microsoft® flag shown), and is displayed only when the browser is loading content. It is removed when the browser is not loading content. FIG. **4** shows display **50** after content has been loaded, during a period when no additional content is being loaded. Graphic element **64** has been removed in FIG. **4** because the current Internet page has been completely loaded.

The temporary graphic element is preferably located in a corner of the content viewing area, and obstructs a portion of the viewing area. The upper right corner is preferred because this position is often blank in Internet documents.

The graphic element is created by opening a conventional window in conjunction with the Window® CE windowing operating environment.

This method of displaying loading status achieves the objective of alerting users during periods of time when content is actually being loaded. It does this without requiring a permanent allocation of screen real estate, thus freeing space for other functions. Although there might be some obstruction of hypermedia content, such obstruction is minor and temporary.

The invention has been described primarily in terms of its visual and functional characteristics. However, the invention also includes a method of browsing a hyperlink resource such as the Internet or some other network or data source having linked hypermedia content. The method includes a steps of loading content from the hyperlink resource in response to user selection of hyperlinks contained in said content, and of displaying the content in a content viewing area. The invention also includes a step of displaying a temporary graphic element over the content viewing area during the loading step. The temporary graphic element is removed when content is no longer being loaded.

Although the invention has been described in language specific to structural features and/or methodological steps, it is to be understood that the invention defined in the appended claims is not necessarily limited to the specific features or steps described. Rather, the specific features and steps are disclosed as preferred forms of implementing the claimed invention.

What is claimed is:

1. A hypermedia browser embodied on a computer-readable medium for execution on an information processing device having a limited display area, wherein the hypermedia browser has a content viewing area for viewing content and is configured to display a temporary graphic element over the content viewing area during times when the browser is loading content, wherein the temporary graphic element is positioned over the content viewing area to obstruct only part of the content in the content viewing area, wherein the temporary graphic element is not content and wherein content comprises data for presentation which is from a source external to the browser.

2. A hypermedia browser as recited in claim **1**, wherein the browser is configured to display the temporary graphic element over the content viewing area only during times when the browser is loading visible content.

3. A hypermedia browser as recited in claim **1**, wherein the temporary graphic element indicates to a user that the browser is loading content.

4. A hypermedia browser as recited in claim **1**, wherein the temporary graphic element disappears when the browser's loading of content is complete to indicate to a user that such loading of content is complete.

5. A hypermedia browser as recited in claim **1**, wherein the temporary graphic element is animated.

6. A hypermedia browser as recited in claim **1**, wherein the hypermedia browser displays the temporary graphic element in a corner of the content viewing area.

7. A hypermedia browser as recited in claim **1**, wherein the hypermedia browser presents the temporary graphic element within a temporary window in a windowing operating environment.

8. A hypermedia browser as recited in claim **1**, wherein:

the temporary graphic element is animated; and

the hypermedia browser presents the temporary graphic element within a temporary window in a windowing operating environment.

9. A hypermedia browser as recited in claim **1**, wherein the temporary graphic element conveys status information of the browser.

10. A hypermedia browser of claim **1**, wherein content is data formatted for presentation which is selected from a group consisting of visible effects of a markup language, visible text of such a markup language, and visible results of a scripting language.

11. A hypermedia browser of claim **1**, wherein content is data formatted for presentation which is selected from a group consisting of HTML, text, SGML, XML, java, XHTML, JavaScript, streaming video, VRML, Active X, Flash. scripting language for the world wide web.

12. An information processing device comprising:

a processor;

a display;

a hypermedia browser executing on the processor to load and display content in a content viewing area on the display;

wherein the hypermedia browser displays a temporary graphic element over the content viewing area during times when the browser is loading visible content;

wherein the temporary graphic element is positioned only over a portion of the content viewing area and obstructs only part of the visible content in the content viewing area; and

wherein the temporary graphic element indicates to a user that the browser is loading content and content comprises data for presentation which is from a source external to the browser.

13. An information processing device as recited in claim **12**, wherein the temporary graphic element is animated.

14. An information processing device as recited in claim **12**, wherein the hypermedia browser displays the temporary graphic element in a corner of the content viewing area.

15. An information processing device as recited in claim **12**, wherein the hypermedia browser displays the temporary graphic element within a temporary window in a windowing operating environment.

16. An information processing device as recited in claim **12**, wherein:

the temporary graphic element is animated; and

the hypermedia browser displays the temporary graphic element within a temporary window in a windowing operating environment.

17. A hypermedia browser of claim **12**, wherein content is data formatted for presentation which is selected from a group consisting of visible effects of a markup language, visible text of such a markup language, and visible results of a scripting language.

18. A hypermedia browser of claim **12**, wherein content is data formatted for presentation which is selected from a group consisting of HTML, text, SGML, XML, java, XHTML, JavaScript, streaming video, VRML, Active X, Flash. scripting language for the world wide web.

19. A method of browsing a hyperlink resource, comprising the following steps:

loading content from the hyperlink resource in response to user selection of hyperlinks contained in said content;

displaying the content in a content viewing area;

displaying a temporary graphic element over the content viewing area during the loading step, wherein the temporary graphic element obstructs only part of the content in the content viewing area;

wherein the loading, the content displaying, and the temporary graphic element displaying steps occur at least partially concurrently; and

wherein content comprises data for presentation which is from a source external to the browser.

20. An information processing device as recited in claim **12**, wherein the temporary graphic element is not content.

21. An information processing device as recited in claim **12**, wherein the temporary graphic element disappears when the browser's loading of content is complete to indicate to a user that such loading of content is complete.

22. A method as recited in claim **19**, wherein the temporary graphic element is not content.

23. A method as recited in claim **19**, wherein the temporary graphic element indicates to a user that the loading step is being performed.

24. A method as recited in claim **19**, further comprising removing the temporary graphic element once the loading step is complete to indicate to a user that the loading step is complete.

25. A method as recited in claim **19**, further comprising an additional step of animating the temporary graphic element.

26. A method as recited in claim **19**, wherein the displaying step includes displaying the temporary graphic element in a corner of the content viewing area.

27. A method as recited in claim **19**, wherein the displaying step includes displaying the temporary graphic element within a temporary window in a windowing operating environment.

28. A method as recited in claim **19**, further comprising an additional step of animating the temporary graphic element, wherein the displaying step includes displaying the temporary graphic element within a temporary window in a windowing operating environment.

29. A computer-readable storage medium containing instructions that are executable for performing the steps recited in claim **19**.

30. A hypermedia browser of claim **19**, wherein content is data formatted for presentation which is selected from a group consisting of visible effects of a markup language, visible text of such a markup language, and visible results of a scripting language.

31. A hypermedia browser of claim **19**, wherein content is data formatted for presentation which is selected from a group consisting of HTML, text, SGML, XML, java, XHTML, JavaScript, streaming video, VRML, Active X, Flash. scripting language for the world wide web.

32. A method of indicating a content "load status" of a hypermedia browser having a content viewing area for viewing content, the method comprising:

displaying loaded content within the content viewing area of a screen of a hypermedia browser, the screen being without a "load status" graphic element, wherein a "load status" graphic element indicates a current content load status of the hypermedia browser;

receiving an instruction to load new content into the content viewing area;

loading such new content into the content viewing area; and

while loading, displaying a "load status" graphic element over the content viewing area so that the graphic element obstructs only part of the content in such content viewing area; and

wherein content comprises data for presentation which is from a source external to the browser.

33. A method as recited in claim **32** further comprising, upon completion of the loading, removing the "load status" graphic element to reveal the part of the content in the content viewing area that the graphic element obstructed when the element was displayed.

34. A hypermedia browser of claim **32**, wherein content is data formatted for presentation which is selected from a group consisting of visible effects of a markup language, visible text of such a markup language, and visible results of a scripting language.

35. A hypermedia browser of claim **32**, wherein content is data formatted for presentation which is selected from a group consisting of HTML, text, SGML, XML, java, XHTML, JavaScript, streaming video, VRML, Active X, Flash. scripting language for the world wide web.

36. A computer-readable medium having computer-executable instructions that, when executed by a computer, perform a method of indicating a content "load status" of a hypermedia browser having a content viewing area for viewing content, the method comprising:

displaying loaded content within the content viewing area of a screen of a hypermedia browser, the screen is without a "load status" graphic element, wherein a "load status" graphic element indicates a current content load status of the hypermedia browser;

receiving an instruction to load new content into the content viewing area;

loading such new content into the content viewing area; and

while loading, displaying a "load status" graphic element over the content viewing area so that the graphic element obstructs only part of the content in such content viewing area; and

wherein content comprises data for presentation which is from a source external to the browser.

37. A hypermedia browser of claim **36**, wherein content is data formatted for presentation which is selected from a group consisting of visible effects of a markup language, visible text of such a markup language, and visible results of a scripting language.

38. A hypermedia browser of claim **36**, wherein content is data formatted for presentation which is selected from a group consisting of HTML, text, SGML, XML, java, XHTML, JavaScript, streaming video, VRML, Active X, Flash. scripting language for the world wide web.

39. A computer-readable medium as recited in claim **36** further having additional computer-executable instructions that perform a method comprising, upon completion of the loading, removing the "load status" graphic element to reveal the part of the content in the content viewing area that the graphic element obstructed when the element was displayed.

40. An information processing device comprising:

a processor;

a display;

a hypermedia browser executing on the processor to load and display content in a content viewing area on the display;

wherein the hypermedia browser is configured to operate in a content-loading mode and a content-loaded mode;

in the content-loaded mode, the hypermedia browser displays loaded content in the content viewing area and no "load status" graphic element is displayed, wherein absence of such "load status" graphic element indicates that the browser is in the content-loaded mode;

in the content-loading mode, the hypermedia browser loads content, displays such content in the content

viewing area as it loads, and displays a "load status" graphic element over the content view area obstructing part of the content displayed in the content viewing area, wherein presence of such "load status" graphic element indicates that the browser is in the content-loading mode; and

wherein content comprises data for presentation which is from a source external to the browser.

41. A hypermedia browser of claim **40**, wherein content is data formatted for presentation which is selected from a group consisting of visible effects of a markup language, visible text of such a markup language, and visible results of a scripting language.

42. A hypermedia browser of claim **40**, wherein content is data formatted for presentation which is selected from a group consisting of HTML, text, SGML, XML, java, XHTML, JavaScript, streaming video, VRML, Active X, Flash. scripting language for the world wide web.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 6,339,780 B1
DATED : January 15, 2002
INVENTOR(S) : Shell et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 5,
Line 15, change "steps" to -- step --.

Signed and Sealed this

Seventeenth Day of September, 2002

Attest:

Attesting Officer

JAMES E. ROGAN
Director of the United States Patent and Trademark Office

Exhibit E

To

Joint Claim Chart

US007411582B2

(12) **United States Patent**
Toepke et al.

(10) **Patent No.: US 7,411,582 B2**
(45) **Date of Patent: *Aug. 12, 2008**

(54) **SOFT INPUT PANEL SYSTEM AND METHOD**

(75) Inventors: **Michael G. Toepke**, Bellevue, WA (US); **Jeffrey R. Blum**, Seattle, WA (US); **Kathryn L. Parker**, Fall City, WA (US)

(73) Assignee: **Microsoft Corporation**, Redmond, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **10/989,877**

(22) Filed: **Nov. 15, 2004**

(65) **Prior Publication Data**

US 2005/0088421 A1 Apr. 28, 2005

Related U.S. Application Data

(63) Continuation of application No. 10/072,111, filed on Feb. 8, 2002, now Pat. No. 6,819,315.

(51) **Int. Cl.**
G06F 3/041 (2006.01)

(52) **U.S. Cl.** **345/173**; 345/179; 345/905; 715/825

(58) **Field of Classification Search** 345/173, 345/179, 347, 762, 156, 901, 905, 87; 715/808, 715/825–829; 395/340
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,058,046	A	10/1991	Lapeyre
5,128,672	A	7/1992	Kaehler
5,252,951	A	10/1993	Tannenbaum et al.

(Continued)

FOREIGN PATENT DOCUMENTS

EP	0464712	1/1992

(Continued)

OTHER PUBLICATIONS

"Function-independent Approach to Driving Soft Keyboards," IBM Technical Disclosure Bulletin, vol. 33, No. 4, pp. 159-161 (Sep. 1, 1990).

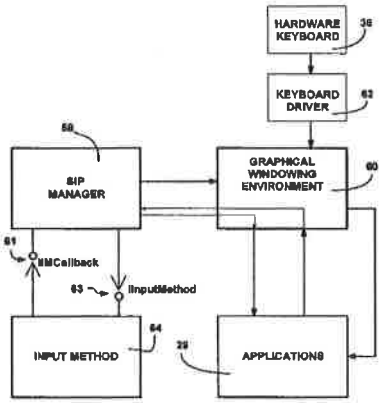
(Continued)

Primary Examiner—Richard Hjerpe
Assistant Examiner—Kimnhung Nguyen

(57) **ABSTRACT**

A method and system for receiving user input data into a computer system having a graphical windowing environment. A touch-sensitive display screen for displaying images and detecting user activity is provided. A management component connects to the graphical windowing environment to create an input panel window for display on the screen. An input method which may be a COM object is selected from multiple input methods available, and installed such that the input method can call functions of the management component. Each input method includes a corresponding input panel, such as a keyboard, which it draws in the input panel window. When the user taps the screen at the input panel, the input method calls a function of the management component to pass corresponding input information appropriate information such as a keystroke or character to the management component. In response, the management component communicates the user data to the graphical windowing environment as a message, whereby an application program receives the message as if the message was generated on a hardware input device.

31 Claims, 8 Drawing Sheets



U.S. PATENT DOCUMENTS

RE34,476	E		12/1993	Norwood
5,276,794	A		1/1994	Lamb, Jr.
5,517,578	A		5/1996	Altman et al.
5,528,743	A		6/1996	Tou et al.
5,574,482	A		11/1996	Niemeier
5,596,702	A		1/1997	Stucka et al. 395/40
5,644,339	A		7/1997	Mori et al. 345/173
5,748,512	A		5/1998	Vargas
5,760,773	A	*	6/1998	Berman et al. 715/808
5,777,605	A		7/1998	Yoshinobu et al.
5,781,181	A		7/1998	Yanai et al.
5,818,425	A		10/1998	Want et al.
5,838,302	A		11/1998	Kuriyama et al.
5,914,707	A	*	6/1999	Kono 345/173
5,936,614	A		8/1999	An et al.
5,956,423	A		9/1999	Frink et al. 382/187
6,008,799	A		12/1999	Van Kleeck
6,018,335	A		1/2000	Onley
6,031,525	A		2/2000	Perlin
6,069,628	A		5/2000	Farry et al. 345/348
6,094,197	A		7/2000	Buxton et al.
6,295,052	B1		9/2001	Kato et al.
6,819,315	B2	*	11/2004	Toepke et al. 345/173

FOREIGN PATENT DOCUMENTS

JP	01-191226	8/1989
JP	06-324806	11/1994
JP	08-22385	1/1996
JP	08-328805	12/1996
WO	WO 9209944	6/1992

OTHER PUBLICATIONS

"Soft Adaptive Follow-Finger Keyboard for Touch-Screen Pads," IBM Technical Disclosure Bulletin, vol. 36, No. 11, pp. 5-7, (Nov. 1, 1993).

Kano, Nadine, *Developing International Software for Windows 95 and Windows NT*, Chapter 7, Appendix N and Appendix O, Microsoft Press, pp. 202-229, 553-556, 557-563.

International Search Report in Corresponding PCT Application No. PCT/US98/26683.

* cited by examiner

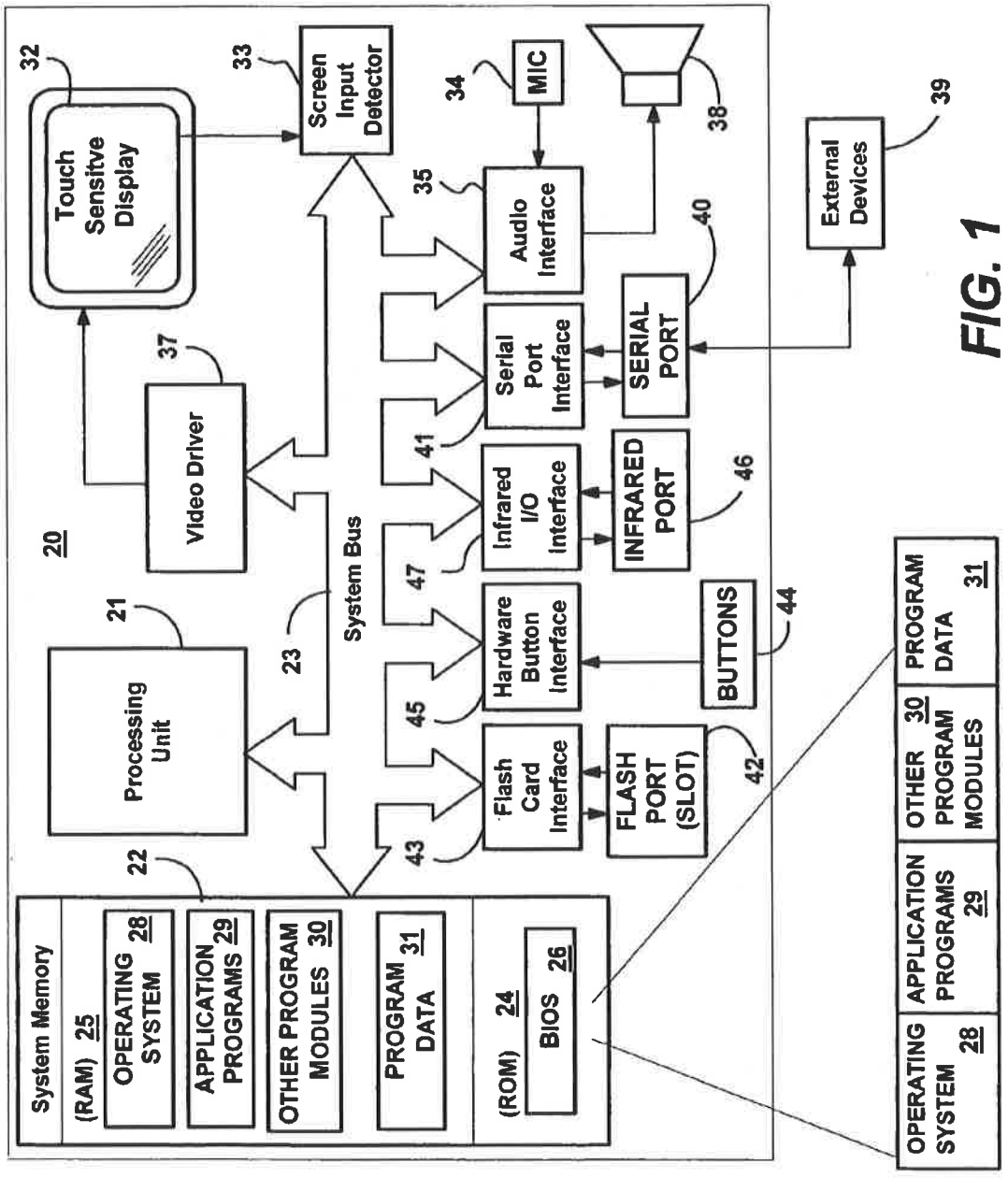

System Memory
(RAM) 25
OPERATING SYSTEM 28
APPLICATION PROGRAMS 29
OTHER PROGRAM MODULES 30
PROGRAM DATA 31
(ROM) 24
BIOS 26
22
20
21
Processing Unit
23
System Bus
Video Driver
37
32
Touch Sensitive Display
33
Screen Input Detector
34
MIC
35
Audio Interface
38
External Devices
39
40
SERIAL PORT
Serial Port Interface
41
Infrared I/O Interface
INFRARED PORT
46
47
Hardware Button Interface
BUTTONS
44
45
Flash Card Interface
FLASH PORT (SLOT)
42
43
OPERATING SYSTEM 28
APPLICATION PROGRAMS 29
OTHER PROGRAM MODULES 30
PROGRAM DATA 31

FIG. 1

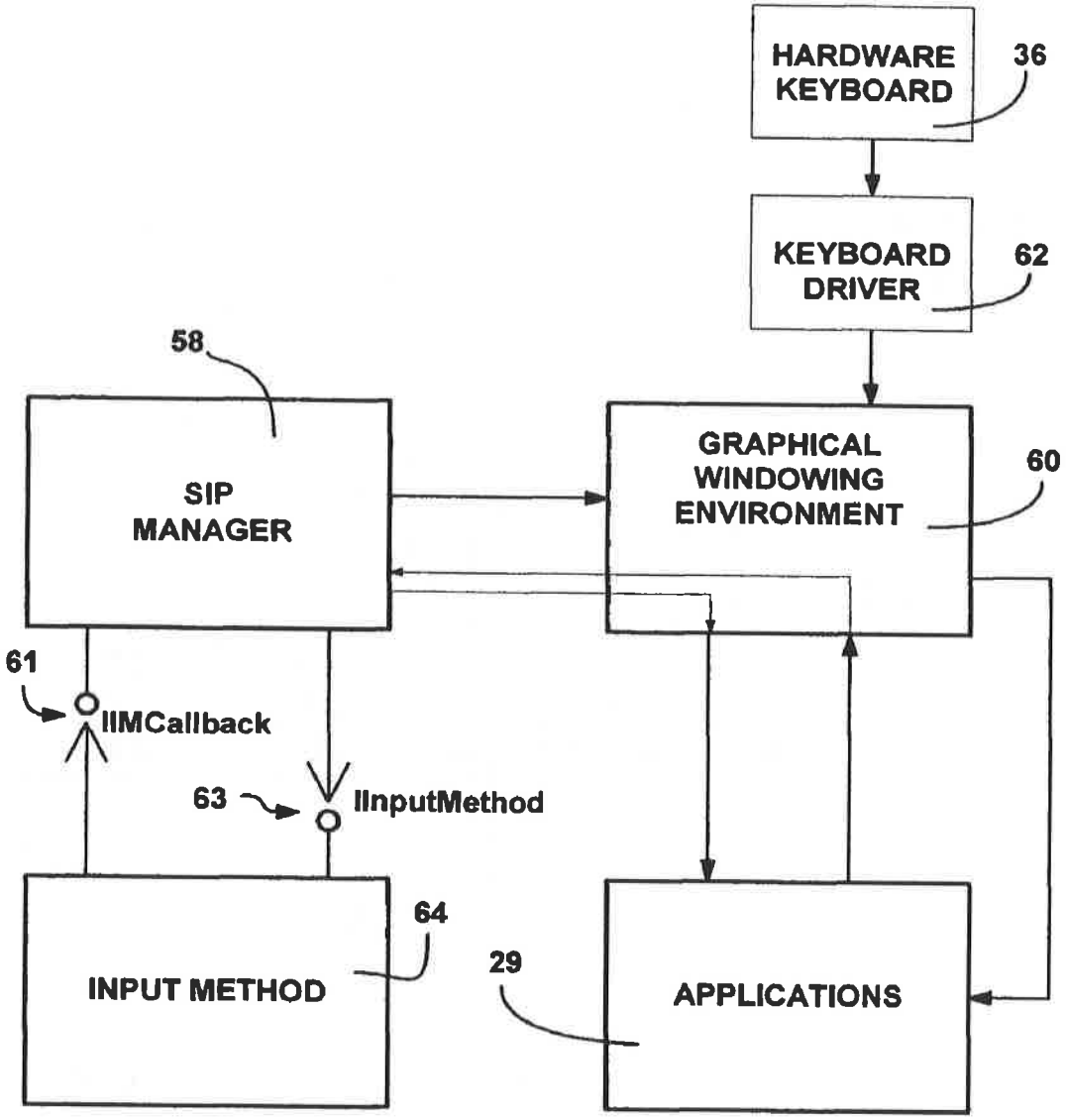

HARDWARE KEYBOARD
36
KEYBOARD DRIVER
62
58
SIP MANAGER
GRAPHICAL WINDOWING ENVIRONMENT
60
61
IIMCallback
63
IInputMethod
64
INPUT METHOD
29
APPLICATIONS

FIG. 2

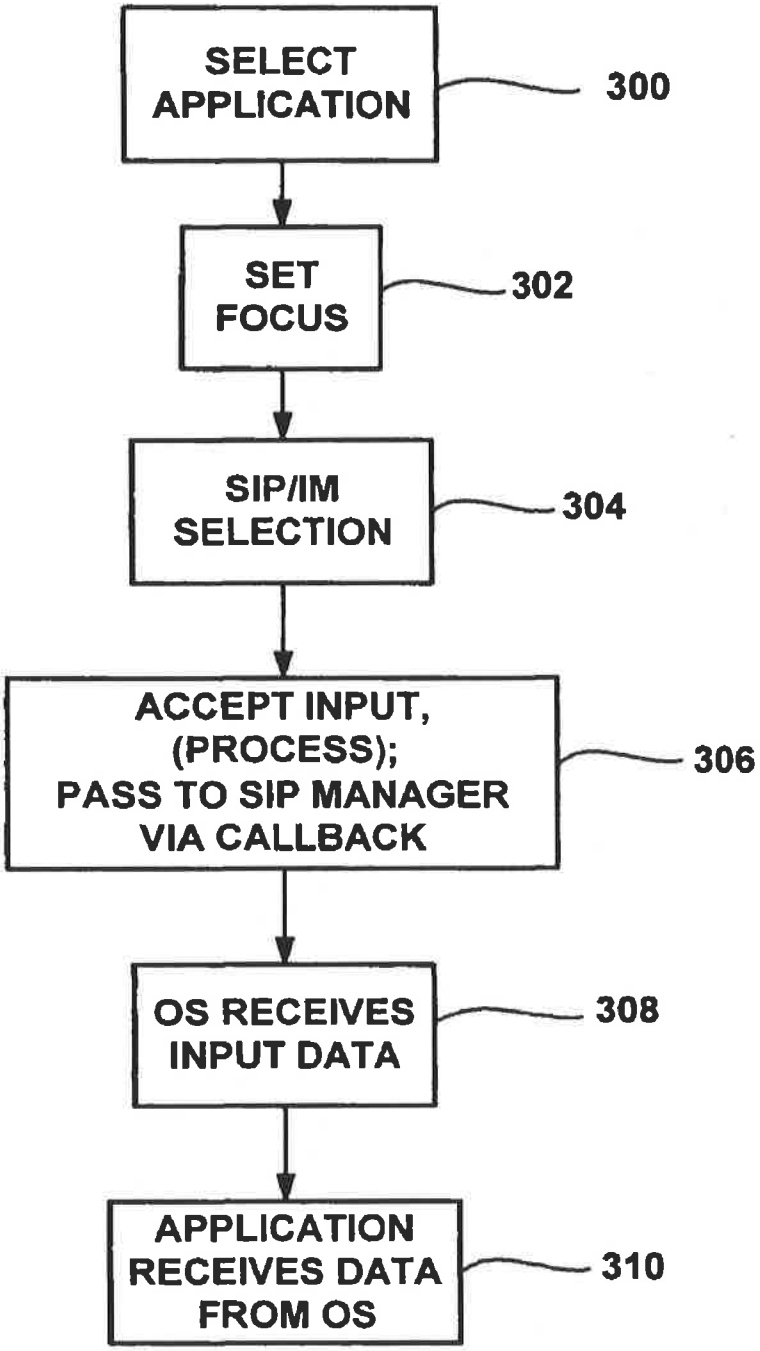

SELECT APPLICATION
300
SET FOCUS
302
SIP/IM SELECTION
304
ACCEPT INPUT, (PROCESS); PASS TO SIP MANAGER VIA CALLBACK
306
OS RECEIVES INPUT DATA
308
APPLICATION RECEIVES DATA FROM OS
310

FIG. 3

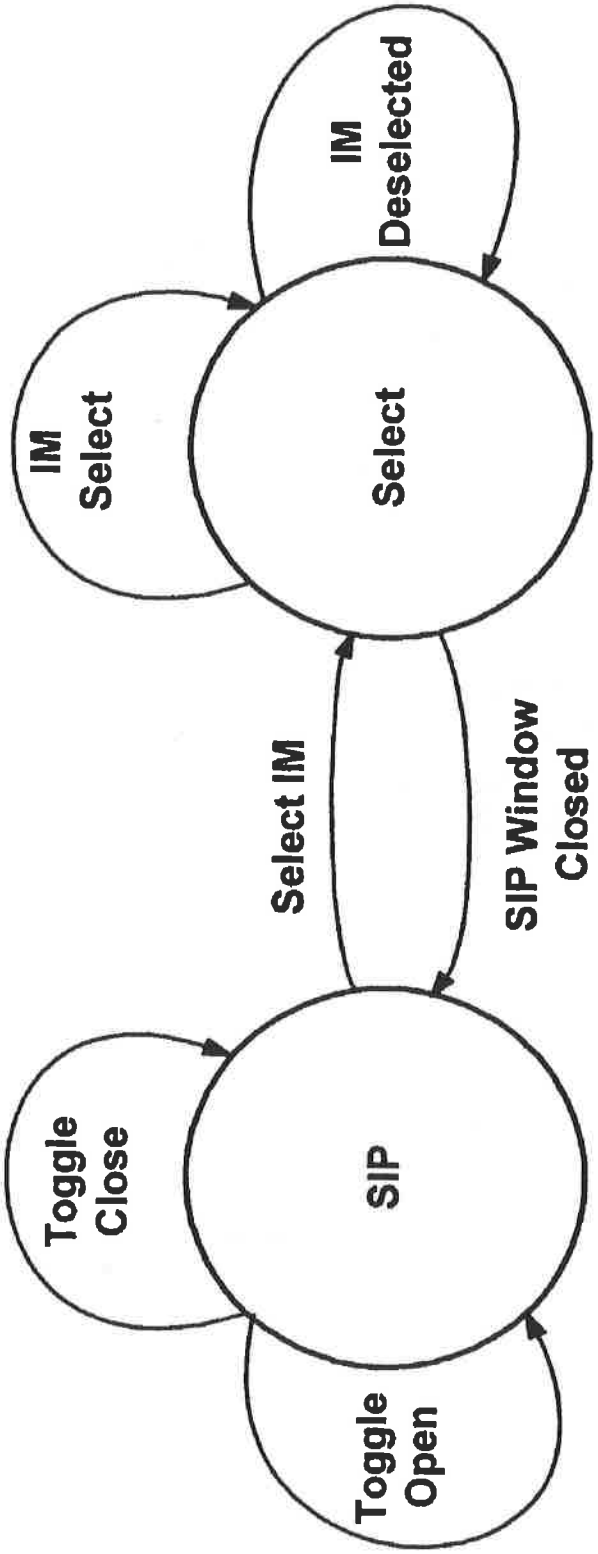

IM Deselected
IM Select
Select
Select IM
SIP Window Closed
Toggle Close
SIP
Toggle Open

FIG. 4

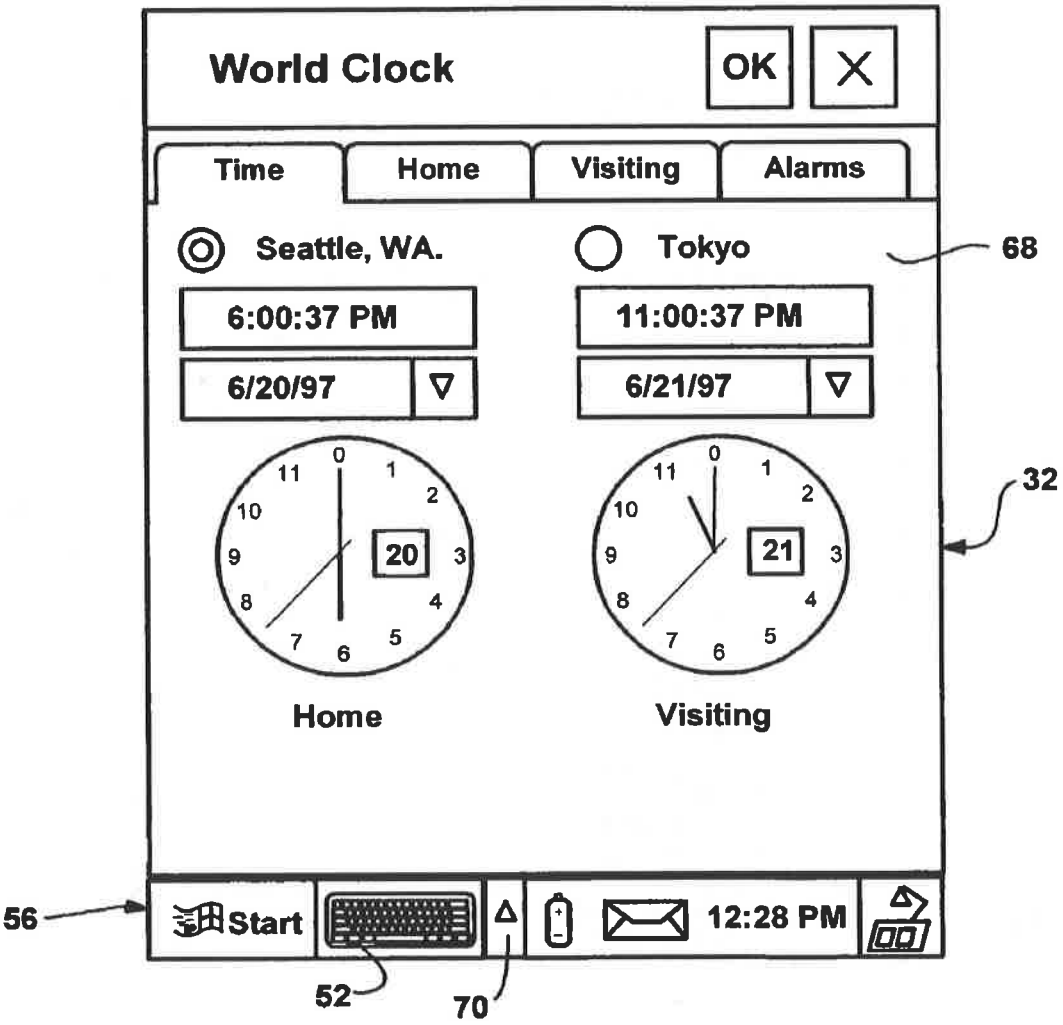

World Clock
OK
Time
Home
Visiting
Alarms
Seattle, WA.
Tokyo
68
6:00:37 PM
11:00:37 PM
6/20/97
6/21/97
32
Home
Visiting
56
Start
12:28 PM
52
70

FIG. 5

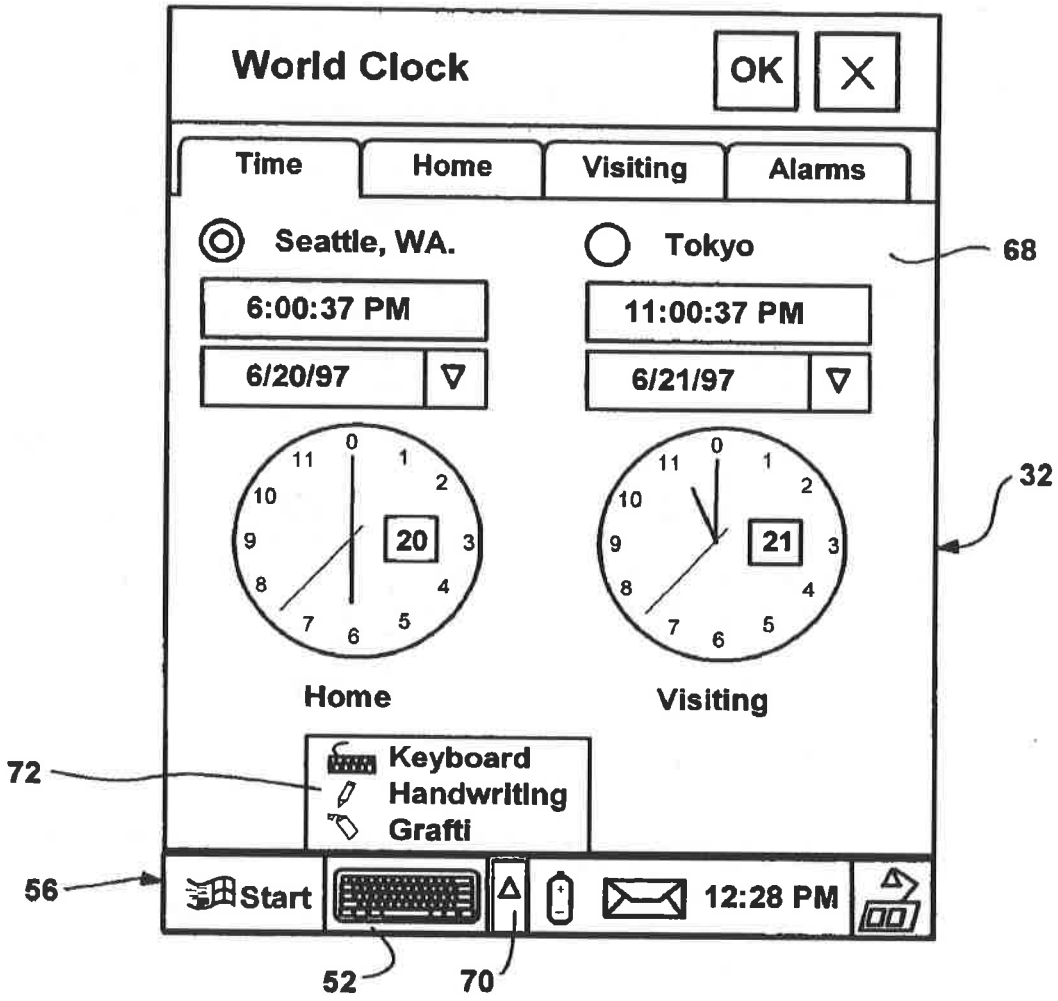

World Clock
OK
Time
Home
Visiting
Alarms
Seattle, WA.
Tokyo
68
6:00:37 PM
11:00:37 PM
6/20/97
6/21/97
32
Home
Visiting
72
Keyboard
Handwriting
Grafti
56
Start
12:28 PM
52
70

FIG. 6

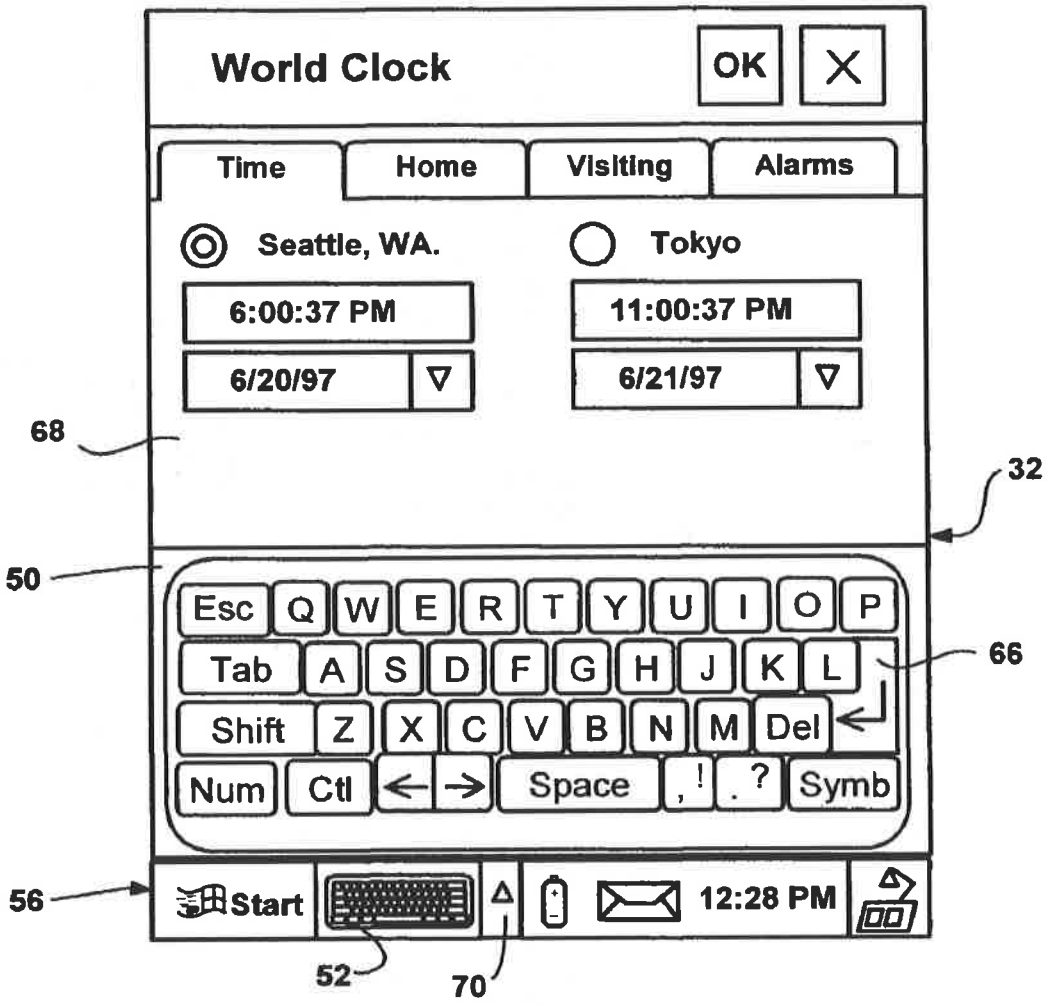

World Clock
OK
X
Time
Home
Visiting
Alarms
Seattle, WA.
Tokyo
6:00:37 PM
11:00:37 PM
6/20/97
6/21/97
68
32
50
Esc Q W E R T Y U I O P
Tab A S D F G H J K L
66
Shift Z X C V B N M Del
Num Ctl Space , ! . ? Symb
56
Start
12:28 PM
52
70

FIG. 7

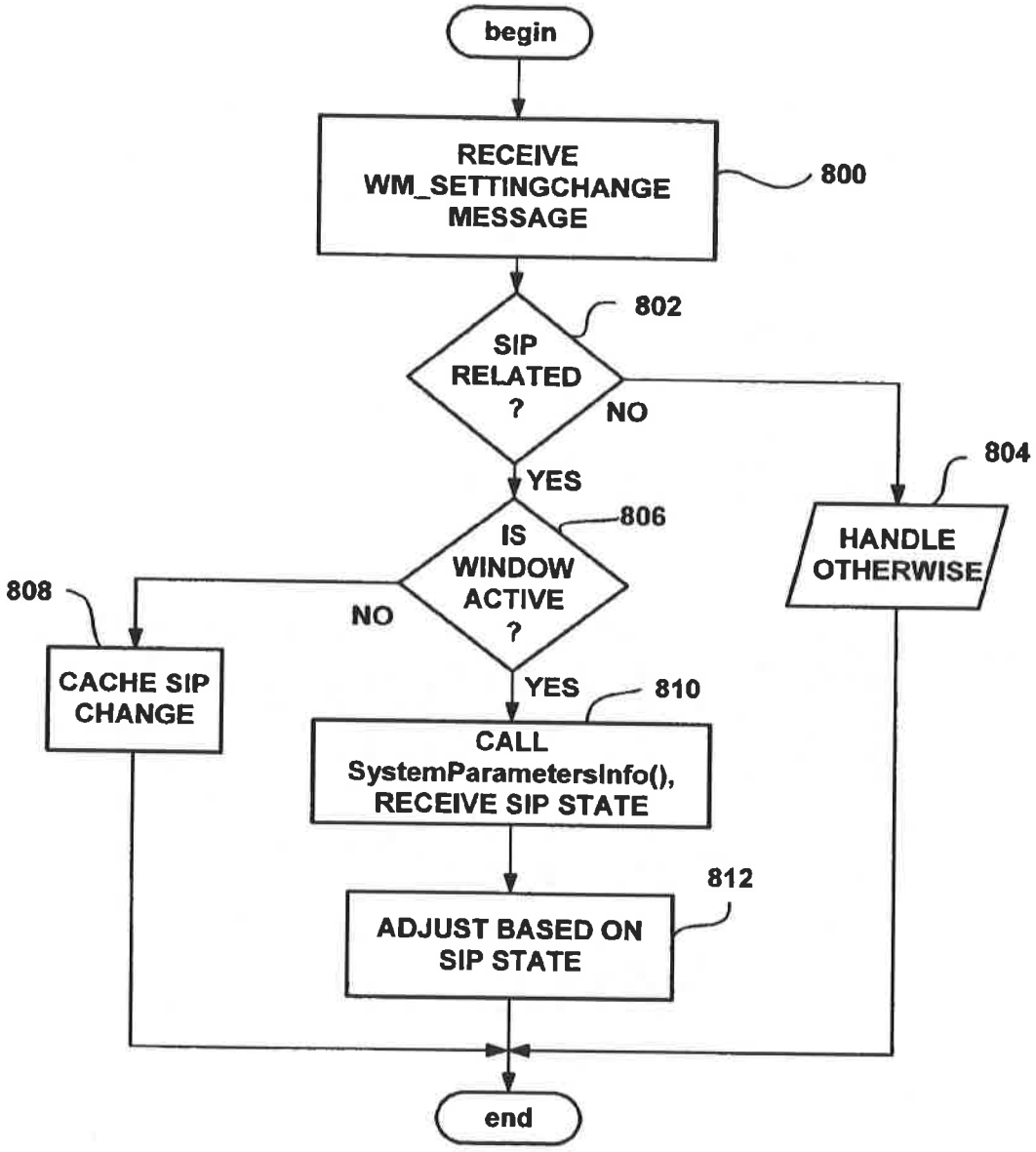

begin
RECEIVE WM_SETTINGCHANGE MESSAGE
800
802
SIP RELATED ?
NO
YES
804
HANDLE OTHERWISE
806
IS WINDOW ACTIVE ?
808
NO
CACHE SIP CHANGE
YES
810
CALL SystemParametersInfo(), RECEIVE SIP STATE
812
ADJUST BASED ON SIP STATE
end

FIG. 8

SOFT INPUT PANEL SYSTEM AND METHOD

CROSS-REFERENCE TOo RELATED APPLICATION

This is a continuation of U.S. patent application Ser. No. 10/072,111 filed Feb. 8, 2002, which is a continuation of U.S. patent application Ser. No. 08/991,277 filed Dec. 16, 1997.

FIELD OF THE INVENTION

The invention relates generally to computer systems, and more particularly to the input of data into a computer system.

BACKGROUND OF THE INVENTION

Small, mobile computing devices such as personal desktop assistants including hand-held and palm-top computers and the like are becoming important and popular user tools. In general, they are becoming small enough to be extremely convenient while consuming less and less battery power, and at the same time becoming capable of running more and more powerful applications.

Although such devices continue to shrink in size, size limitations are being reached as a result of human limitations. For example, a full character keyboard that enables user data input cannot be so small that human fingers cannot depress the individual keys thereon. As a result, the size of such devices (e.g., palm-top computers) has become limited to that which can accommodate a full character keyboard for an average user.

One solution to reducing the size of the portion of the device that receives user input is to provide a touch-sensitive display, and thereby substantially eliminate the need for a physical keyboard. To this end, an application program such as a word processor displays a keyboard, whereby the user enters characters by touching the screen at locations corresponding to the displayed keys. Of course, touch screen devices can also be used simultaneously with devices having a physical keyboard, whereby characters can also be entered by manually pressing the keys of the physical keyboard.

While a touch-screen device serves to provide a suitable means of user data entry, the data entry panel is typically part of the application program, i.e., each application needs to develop its own touch-sensitive interface. As a result, a substantial amount of duplication takes place. For example, both the word processor and a spreadsheet program require alphanumeric keyboard input, whereby each provides its own touch-screen keyboard interface. Other types of programs, such as a calculator program, need a numeric keypad with additional keys representing mathematical operations. This makes each program larger, more complex and consumes computer system resources.

Alternatively, the operating system can supply all the virtual keyboards and thus eliminate the redundancy, however this limits applications to using only those virtual keyboards supplied by the operating system. Newer applications (e.g., those added by plug-in modules) are unable to provide an input mechanism that is more tailored to its particular needs. For example, a new paintbrush program may need its own graphical input screen. In sum, there is a tradeoff between flexibility and efficiency that is inherent with present user data input mechanisms.

OBJECTS AND SUMMARY OF THE INVENTION

Accordingly, it is an object of the present invention to provide an improved method system for entering user data into a computer system.

Another object of the present invention is to provide the method and system for user data entry that is both efficient and flexible.

In accomplishing those objects, it is a related object to provide a method and system of the above kind that functions with touch-sensitive input mechanisms.

Yet another object is to provide a method and system as characterized above that enables a plurality of applications to receive user input from a common input method.

A related object is to provide a method and system that enables selection of one or more input methods for each application from among a set of interchangeable input methods.

Yet another object is to provide such a method and system that is cost-effective, reliable, extensible and simple to implement.

Briefly, the present invention provides a method and system for receiving user data input into a computer system, such as a computer system having a graphical windowing environment. The invention may utilize a touch-sensitive display screen for displaying images and detecting user contact therewith (or proximity thereto). A management component operatively connected to the graphical windowing environment creates an input panel window for display on the screen. An input method is selected from among a plurality of such input methods and installed, whereby the input method can call functions of the management component. Each input method includes a corresponding input panel, such as a keyboard, which it draws in the input panel window. When user data is received via the input panel, the input method calls a function of the management component to pass the user data thereto, and in response, the management component communicates the user data to the graphical windowing environment such as in a windows message. An application program receives the message, such as corresponding to a keystroke, as if the message was generated on a hardware keyboard.

Other objects and advantages will become apparent from the following detailed description when taken in conjunction with the drawings, in which:

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram representing a computer system into which the present invention may be incorporated;

FIG. **2** is a block diagram representing various components and connections therebetween for implementing interchangeable input panels according to an aspect of the present invention;

FIG. **3** is a flow diagram generally representing a process for getting user input from a selected input method to a selected application in accordance with one aspect of the present invention;

FIG. **4** is a state diagram generally representing SIP selection states;

FIG. **5** represents a display on a touch-sensitive display screen on an exemplary computing device;

FIG. **6** represents a display on a touch-sensitive display screen on an exemplary computing device providing the ability to select from among interchangeable input panels in accordance with the present invention;

FIG. **7** represents a display on a touch-sensitive display screen wherein a keyboard has been selected as an input panel in accordance with the present invention; and

FIG. **8** is a flow diagram representing the general steps taken in response to a change in SIP status.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Exemplary Operating Environment

FIG. **1** and the following discussion are intended to provide a brief, general description of a suitable computing environment in which the invention may be implemented. Although not required, the invention will be described in the general context of computer-executable instructions, such as program modules, being executed by a hand-held computing device such as a personal desktop assistant. Generally, program modules include routines, programs, objects, components, data structures and the like that perform particular tasks or implement particular abstract data types.

Moreover, those skilled in the art will appreciate that the invention may be practiced with other computer system configurations, including palm-top, desktop or laptop personal computers, mobile devices such as pagers and telephones, multi-processor systems, microprocessor-based or programmable consumer electronics, network PCs, minicomputers, mainframe computers and the like. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

With reference to FIG. **1**, an exemplary system for implementing the invention includes a general purpose computing device in the form of a hand-held personal computing device **20** or the like, including a processing unit **21**, a system memory **22**, and a system bus **23** that couples various system components including the system memory to the processing unit **21**. The system bus **23** may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. The system memory includes read-only memory (ROM) **24** and random access memory (RAM) **25**. A basic input/output system **26** (BIOS), containing the basic routines that help to transfer information between elements within the hand-held computer **20**, such as during start-up, is stored in the ROM **24**.

A number of program modules are stored in the ROM **24** and/or RAM **25**, including an operating system **28** (preferably Windows CE), one or more application programs **29**, other program modules **30** and program data **31**. A user may enter commands and information into the hand-held computer **20** through input devices such as a touch-sensitive display screen **32** with suitable input detection circuitry **33**. Other input devices may include a microphone **34** connected through a suitable audio interface **35** and a physical (hardware) keyboard **36** (FIG. **2**). The output circuitry of the touch-sensitive display **32** is also connected to the system bus **23** via video driving circuitry **37**. In addition to the display **32**, the device may include other peripheral output devices, such as at least one speaker **38** and printers (not shown).

Other external input or output devices **39** such as a joystick, game pad, satellite dish, scanner or the like may be connected to the processing unit **21** through an RS-232 or the like serial port **40** and serial port interface **41** that is coupled to the system bus **23**, but may be connected by other interfaces, such as a parallel port, game port or universal serial bus (USB). The hand-held device **20** may further include or be capable of connecting to a flash card memory (not shown) through an appropriate connection port (e.g., slot) **42** and interface **43**. A number of hardware buttons **44** such as switches, buttons (e.g., for switching application) and the like may be further provided to facilitate user operation of the device **20**, and are also connected to the system via a suitable interface **45**. An infrared port **46** and corresponding interface/driver **47** are provided to facilitate communication with other peripheral devices, including other computers, printers, and so on (not shown). It will be appreciated that the various components and connections shown are exemplary and other components and means of establishing communications links may be used.

Soft Input Panel

The soft input panel architecture is primarily designed to enable character, key-based and other user data input via the touch screen **32** of the device **20** rather than a physical keyboard **36**. However, as can be appreciated, a given computer system **20** may optionally and additionally include a physical keyboard, as represented by the dashed box **36** of FIG. **2**. Moreover, as will become apparent, the "soft input panel" need not be an actual touch-sensitive panel arranged for directly receiving input, but may alternatively operate via another input device such as the microphone **34**. For example, spoken words may be received at the microphone **34**, recognized, and displayed as text in an on-screen window, i.e., a soft input panel.

FIG. **2** shows a block diagram implementing the SIP architecture in accordance with one aspect of the present invention. The computer system **20** includes an operating system (OS) **28** such as the graphical windowing environment **60**. Such a graphical windowing environment **60** is generally operational to receive user input through a variety of devices including the keyboard **36**, a mouse (not shown), a digitizer (not shown) and so on. In turn, the graphical windowing environment **60** may provide such user input to an application having "input focus," typically in the form of a keyboard character event. Note that a number of applications **29** may be executable by the computer system, however one application that is currently running is said to have "input focus" and receive the input.

In accordance with one aspect of the present invention, the present architecture employs a SIP manager **58** to provide a single and flexible interface for a plurality of different input methods **64**. In general, the SIP manager **58** provides keystrokes from a selected input method **64** to the graphical windowing environment **60** (e.g., the Windows CE operating system **28**). Once received, the graphical windowing environment **60** sends information corresponding to the user input data to an application **29** (i.e., the application whose window currently has input focus) in the form of that keystroke, mouse or other message placed in the message queue of the application's window. The passing of such messages is well known in Windows programming and is described in "*Programming Windows* 95," Charles Petzold, Microsoft Press (1996), hereby incorporated by reference. As a result, any application capable of handling keyboard input may be used with any appropriately-configured input method **64**. Indeed, if an optional keyboard **36** is present, keystrokes are directly provided by a keyboard driver **62** to the graphical windowing environment **60**, whereby appropriate keystrokes are likewise placed in the message queue of the active application's window without the application being provided with information as to the source.

Input methods **64** may include, for example, various different displayable keyboards, (soft keyboards), a calculator, a formula and/or equation editor, chemical symbol template, voice recognition, handwriting recognition, shorthand symbol recognition (such as "Graffiti"), or other application-optimized input methods (e.g. a barcode reader). The SIP manager **58** provides a user interface for permitting a user to toggle a SIP window (panel) **50** (FIG. **7**) between an opened and closed state, as described in more detail below. The SIP manager **58** also provides a user interface enabling user selection from a displayable list of available input methods. A user interacting with the user interface may select an input method **64**, and in response, the SIP manager **58** loads and calls the selected input method **64**. In a preferred embodiment, each of the input methods communicates with the SIP manager **58** through a COM (Component Object Model) interface shown as IIMCallback **61** and IInputmethod **63**. A COM object comprises a data structure having encapsulated methods and data that are accessible through specifically defined interfaces. A detailed description of COM objects is provided in the reference entitled "Inside OLE," second edition, Kraig Brockschmidt (Microsoft Press), hereby incorporated by reference.

Generally, when the SIP window **50** is toggled between on/off by a user, as will be described in more detail below, the SIP manager **58** informs the selected input method **64** to correspondingly open/close the SIP window **50** through the IInputmethod mechanism **63**. When a new input method is selected, the SIP manager **58**, through the mechanism **63**, informs any of the previously selected input methods to exit, and loads the newly selected input method. The mechanism **63** may also be utilized by the SIP manager **58** to obtain information specific to a selected input method, as also described in detail below.

The selected input method **64** may also communicate information to the SIP manager **58** via the IIMCallback mechanism **61**, such as which character or characters were entered by a user, irrespective of whether the character or characters are generated through keyboard selection, handwriting recognition, voice recognition, a formula editor, calculator or the like. Such character input is generally passed to the SIP manager **58**, preferably received as (or converted to) a Unicode character (for Windows CE) by the SIP manager **58** and output to the graphical windowing environment **60**. Command key information, such as "Ctrl" on a keyboard, may also be provided by the input method **64** to the SIP manager **58** via interface **61**.

SIP and input method-specific information may also be communicated through the SIP manager **58**, and ultimately to the focused application **29**, when the application is optimized for operating with a SIP (i.e., is "SIP-aware") as described in more detail below.

The system operates as generally represented in the steps of FIG. **3**. Once an application is selected and has focus (steps **300-302**), an input method **64** is selected therefor at step **304**. Note that the input method **64** may be selected by the user, or a default input method may be selected for use with a particular application. Additionally, the input method **64** may be one that remains after having been selected for a previous application, i.e., a particular input method stays the same as the user switches between various applications. In any event, the input method **64** displays a SIP window **50** when selected.

As the user inputs data at step **306**, appropriate data is passed to the SIP manager **58** via the IIMCallback mechanism **61**, described below. Note that the input method **64** may first process the received data at step **306**. By way of example, one particular input method **64** may convert barcode symbols to Unicode characters representing digits, another input method may convert mathematical entries into a Unicode result (e.g., an entry of '3+6=' sends a '9' to the SIP manager **58**), while yet another may be an equation editor (e.g., the characters "Sqrt" are converted into a single Unicode value representing a square root symbol). After any such processing, the input method **64** passes those digits to the SIP manager **58**, which in turn passes those digits to the graphical windowing environment **60**. The application receives the character data from the graphical windowing environment **60** as if the user had entered those digits on a physical keyboard, regardless of the input method used.

As shown in FIGS. **5-7**, the soft input panel (SIP) functionality of the system collectively includes the visible window **50** (FIG. **7**), a visible SIP button **52**, and various methods and functions (described below). As shown in FIG. **7**, the SIP window **50** is a rectangular area provided by the input method **64** that can be hidden or shown at the user's (or an application program's) request. The visible SIP button **52** is located on a taskbar **56** or the like, and provides a touch-sensitive interface by which the user displays or hides the SIP window **50**. Thus, as represented in the state diagram of FIG. **4**, the window **50** toggles between an open, visible state (FIG. **7**) and a closed, hidden state (FIG. **5**) as the user taps the SIP button **52**. A present design implements a 240 pixel wide by 80 pixel high SIP window **50** that is fixed (docked) on the display **32** at a position just above the taskbar **56**. As will become apparent below, the soft input panel design supports other SIP window **50** sizes or positions.

To this end, the operating system **28** creates a dedicated thread (the SIP manager **58**) that registers itself as a SIP thread with the Windows CE system. The thread creates the SIP window **50**, performs other SIP initialization, and then enters a message loop to respond to messages and user interface activity in the SIP window **50**. The thread also serves to dispatch messages to an Input Method's window, and calls into the Input Method **64** to permit the Input Method **64** to create windows that will respond as special SIP windows.

The SIP manager thread **58** is given special status by the system. For example, windows created by the SIP manager **58** thread are topmost windows, and ordinarily will not be obscured by other windows, except, e.g., when the taskbar **56** is activated in an auto-hide mode while the SIP window **50** is displayed. In this case, the SIP window **50** remains displayed in its current location and the taskbar **56** is displayed on top of the SIP window **50**. More generally, any user interface element for controlling the SIP may (and should) be placed on top of (rather than underneath) the SIP window **50**, whenever the controlling user interface element and the SIP window **50** overlap.

Moreover, when tapped on, the SIP window **50** (and any child windows thereof such as pushbuttons, text entry fields, scrollbars and the like) will not receive the input focus as would conventional program windows. In this manner, the user may interact with the SIP window **50** without changing the system focus. As can be appreciated, changing the system focus each time the user inputs data into the SIP window **50** would be undesirable. The SIP button **52** will also not cause a change of focus for the same reason, i.e., it is undesirable to cause the window with focus to lose focus by tapping on the SIP button **52** to bring out the SIP window **50**.

In accordance with one aspect of the present invention, the SIP system enables the selective installation of a specified Input Method **64**. As generally described above, each Input Method **64** is an interchangeable component by which the user provides character, text or other user data via the touch-screen display (or some other input device). More particu-

larly, the SIP manager **58** preferably exposes a COM interface that enables the selective installation of Input Methods **64**. The Input Method **64** occupies space inside a SIP window **50** created by the system.

Preferably, the Input Method **64** comprises a Component Object Model (COM) object that implements the IInputMethod interface. Notwithstanding, the Input Method **64** and SIP manager **58** can comprise virtually any components capable of communicating with one other through some mechanism, such as by receiving, responding to, and making function calls.

The Input Method **64** is responsible for drawing in the SIP window **50** and responding to user input in the SIP window **50**. Typically, the Input Method **64** will respond to user input and convert that input into characters which are then sent to the SIP manager **58** via exposed SIP functions. By way of example, one Input Method **64** includes a default QWERTY (alpha) keyboard **66** shown in FIG. **7**. More particularly, this Input Method **64** displays an image of the keyboard **66** on the screen **32**, and converts taps on that keyboard **66** (detected as screen coordinates) into characters which are sent to the SIP manager **58** and thereby to the system. Input Methods may be written by application vendors, and are added to the system using COM component installation procedures.

The user interacts with the Input Method **64** manifested in the visible SIP window **50** to create system input. As best represented by the state diagram of FIG. **4** and as shown in FIG. **6**, the user can select a different Input Method by tapping a SIP menu button **70** on the taskbar **56** that provides a pop-up input method list **72** into the SIP window **50**. The user can also select among available Input Methods via a control panel applet (not shown) or the like. The SIP control panel applets communicate with the operating system **28** using the registry and the exposed SIP-aware functionality described below.

As will be described in detail below, the various components cooperate to expose functions, structures, and window messages that enable system applications **29** to respond to changes in the SIP state. An application **29** that uses this functionality to adjust itself appropriately to SIP changes is considered "SIP-aware." Other applications may be SIP-aware yet choose to retain their original size (and thus be partially obscured by the SIP window **50**) when appropriate. Moreover, and as also described below, there are exposed functions that enable applications to programmatically alter the SIP state.

Notwithstanding, applications **29** need not be aware of the SIP system in order to benefit from the present invention. Indeed, one aspect of the present invention is that applications do not ordinarily recognize whether data received thereby originated at a hardware input device such as the keyboard **36** or via user activity (e.g., contact or proximity detected by the screen **32** and detection circuitry **33**) within the soft input panel window **50**. This enables applications to operate with virtually any appropriate input method, irrespective of whether that application is SIP-aware.

Turning to an explanation of the mechanism that facilitates the operation of an Input Method **64** installed by the SIP manager **58**, a SIP-aware application **29** is notified when the SIP window **50** changes state and what the new, current state of the SIP window **50** is. The state includes whether the status of the SIP window **50** is visible or hidden, whether the SIP window **50** is docked or in a floating condition, and the size and position of the SIP window **50**. As shown in the table below, a data structure (SIPINFO) contains this SIP information:

```
Typedef struct {
    DWORD    cbSize
    DWORD    fdwFlags
    RECT     rcVisibleDesktop
    RECT     rcSipRect
    DWORD    dwImDataSize
    Void *pvImData
} SIPINFO;
```

The cbSize field may be filled in by the application program **29** and indicates the size of the SIPINFO structure. This field allows for future enhancements while still maintaining backward compatibility, and indeed, the size of the SIPINFO structure may be used to indicate the version to the components of the system. The fdwFlags field represents the state information of the SIP window **50**, and can be a combination of three flags. A SIPF_ON flag that is set indicates that the SIP window **50** is visible (i.e., not hidden), while a set SIPF_DOC flag indicates the SIP window **50** is docked (i.e. not floating). A set SIPF_LOCKED flag indicates that the SIP window **50** is locked, i.e., the user cannot change its visible or hidden status. Note that a given implementation may not allow floating or locked SIP windows, however the capability is present within the system.

The rcVisibleDesktop field contains a rectangle, in screen coordinates, representing the area of the screen desktop **68** not obscured by the SIP window **50**. If the SIP window **50** is floating (not docked), this rectangle is equivalent to the user-working area. Full-screen applications wishing to respond to SIP window **50** size changes can generally set their window rectangle data structure ("rect") values to this RECT data structure's values. If the SIP window **50** is docked and does not occupy an entire edge (top, bottom, left or right), then this rectangle represents the largest rectangle not obscured by the SIP window **50**. However, the system may provide available desktop space **68** not included in the RECT data structure.

Next, the rcSipRect field contains the rectangle, in screen coordinates, representing the size and location of the SIP Window **50**. Applications **29** will generally not use this information, unless an application **29** wants to wrap around a floating SIP window **50** or a docked SIP window **50** that is not occupying an entire edge.

The dwImDataSize field contains the size of the data pointed to by the PvImData member, which is the next field, i.e., a pointer to the Input Method-specific data. The data are defined by the Input Method **64**.

Whenever the state of the SIP window **50** changes, i.e., a new Input Method has been selected and/or a visibility, docking or size change has occurred, a message, WM_SETTINGCHANGE, is sent to all top-level windows, as generally represented at step **800** of FIG. **8**. In this manner, an application **29** can adjust itself to the new state of the SIP window **50**, such as by adjusting its size in response to this message. To this end, a flag, SPI_SETSIPINFO, is sent with this message to indicate when SIP information has changed, and another flag, SPI_SETCURRENTIM, when the current Input Method has changed. As shown at step **802** of FIG. **8**, the flag is tested to determine if the message is SIP-related or another type of setting change message (whereby it is handled at step **804**). If SIP-related, for performance reasons, the applications that are not currently active in the foreground cache these SIP changes (steps **806-808**). If the application's window is active, the application can adjust its size and/or window (steps **810-812**). For example, as shown in FIGS. **5** and **6**, when the SIP window **50** of FIG. **7** is hidden and an active application

29 notified, the application 29 may use the additional desktop space 68 to display more information such as the analog clock faces. Note that an application 29 that has cached a SIP change when inactive can query the current SIP state when activated to subsequently adjust itself in an appropriate manner in accordance with the information that is returned.

To query the SIP manager 58, another function, SHSipInfo, is provided so that applications 29 can determine information about the SIP window 50 and Input Method 64. In general, if this function succeeds, the return value will be nonzero, while if this function fails, the return value will equal zero and extended error information will be available via a GetLastError( ) call.

The following table sets forth the structure of this call:

```
SHSipInfo (
    UINT uiAction
    UINT uiParam
    PVOID pvParam
    UINT fwinIni
);
```

The uiAction parameter can include the values SIP_SETSIPINFO, SPI_GETSIPINFO, SPI_SETCURRENTIM and SPI_GETCURRENTIM. SIP_SETSIPINFO indicates that pvParam points to a SIPINFO structure (described above). The cbsize, dwImDataSize and pvImDataSize are filled in before calling the SHSipInfo function. In response to this call, the SIPINFO structure is filled in with the current SIP size, state, and visible desktop rectangle. If both dWImDataSize and pvImData are nonzero, the data size and pointer are sent to the Input Method 64. If the Input Method 64 is called but does not provide Input Method-specific data, or the format or size of the data passed in is not in a format recognized by the Input Method 64, then the SHSipInfo function call fails (returns zero). If the size and format are supported by the Input Method 64, the Input Method 64 fills in the buffer that is pointed to by pvImData with the Input Method-specific data. Typically, an application 29 will set the pvImDataSize to zero and pvImData to NULL.

A uiAction of SPI_SETSIPINFO indicates that pvParam points to a SIPINFO structure. The SIP window 50 size and state are set to the values specified in the SIPINFO structure. Before changing a SIP value, the application 29 should first obtain the current SIP state by calling SHSipInfo with SPI_GETSIPINFO, then change whatever specific SIP state values it wishes to change before making the SPI_SETSIPINFO call. The cbSize field is set to the size of the SIP in the structure, and if both pvImDataSize and pvImData are not zero, the data size and pointer are sent to the Input Method 64. The SHSipInfo call fails if the Input Method 64 is called and does not allow setting Input Method-specific data, or if the format or size of the passed data is not in a format recognized thereby. If a size and format are supported by the Input Method 64, the Input Method 64 uses the data to set Input Method-specific information. Typically, an application will set the pvImDataSize to zero and pvImData to NULL.

SPI_SETCURRENTIM indicates that pvParam points to a CLSID structure which specifies the CLSID of the Input Method 64 to which the SIP will switch. If the CLSID is not valid, or if the specified Input Method 64 cannot be loaded, the call fails (return value equals zero) and a default Input Method 64 (e.g., the QWERTY-like keyboard 66) is loaded.

Lastly, a uiAction of SPI_GETCURRENTIM indicates that pvParam points to a CLSID structure that receives the CLSID of the currently selected Input Method 64.

The IInputMethod Interface

IInputMethod is the interface implemented by the Input Method 64 components. The SIP manager 58 calls the methods of this interface to notify the Input Method 64 of state changes, and request action and information from the Input Method 64. In general, if the called method succeeds, a success is returned, and conversely, if the method fails, a failure result is returned. The following table sets forth the method calls available in this IInputMethod interface:

```
Interface IinputMethod : Iunknown
{
  HRESULT Select ( [in] HWND hwndSip );
  HRESULT Deselect( void );
  HRESULT Showing ( void );
  HRESULT Hiding ( void );
  HRESULT GetInfo ( [out] IMINFO *pimi );
  HRESULT ReceiveSipInfo ( [in] SIPINFO *psi );
  HRESULT RegisterCallback ( [in] IIMCallback* pIMCallback );
  HRESULT GetImData ( [in] DWORD dwSize, [out] LPVOID
  pvImData );
  HRESULT SetImData ( [in] DWORD dwSize, [in] LPVOID
  pvImData );
  HRESULT UserOptionsDlg ( [in] HWND hwndParent );
}
```

An Input Method 64 will ordinarily receive a Select( ), GetInfo( ), ReceiveSipInfo( ) and Register Callback( ) method call, in sequence, before rendering the SIP window 50 space or responding to user actions. When the SIP window 50 is displayed (i.e., turned on), Showing( ) will be called by the SIP manager 58, after which the Input Method 64 issues a WM_PAINT message to render the SIP window 50.

The Select( ) method is called when the Input Method 64 has been selected into the SIP. The Input Method 64 generally performs any desired initialization in response to this call. The Input Method is responsible for drawing the entire client area of the SIP window 50, and thus ordinarily creates its windows and imagelists (collections of displayable bitmaps such as customized icons) in response to this call. For example, the window handle of the SIP window 50 is provided to the Input Method 64 as a parameter accompanying this Select( ) method call, and the Input Method normally creates a child window of this SIP window 50. The Input Method 64 is also provided with a pointer to a value, which is set to nonzero by the Input Method 64 if the method call is successful or zero if not successful.

The Deselect( ) method is called when the Input Method 64 has been selected out of the SIP. The Input Method's window should be destroyed in response to this call, and the Input Method 64 will typically perform any other cleanup at this time.

The Showing( ) method will cause the SIP window 50 to be shown upon return from the call. Note that the SIP window 50 is not visible prior to this call, and that once the SIP window 50 is shown, this window and its children will receive paint messages. Conversely, the Hiding( ) method hides the SIP window 50 upon return from the call. Accordingly, the Showing( ) and Hiding( ) methods are used to toggle the SIP window 50 between its open and closed states.

The GetInfo( ) method is called when the system is requesting information about the Input Method 64. The information requested includes flags indicating any special properties of the Input Method 64, the handles of two imagelists which contain masked bitmaps that are to be displayed on the SIP button 52 when that Input Method 64 is active, indices into the specified imagelists, and a rectangle indicating the preferred

size and placement of the Input Method **64**. The call includes a parameter, pimi, which is a pointer to a data structure (IMINFO) that the Input Method **64** should fill in with appropriate data. The call also provides a pointer to a value that the Input Method should set to nonzero to indicate success and zero to indicate failure. More particularly, the IMINFO data structure is represented in the following table:

```
Typedef struct {
    DWORD cbSize;
    HIMAGELIST hImageNarrow;
    HIMAGELIST hImageWide;
    Int iNarrow;
    Int iWide;
    DWORD fdwFlags;
    Rect rcSipRect;
} IMINFO;
```

The cbSize field contains the size of the IMINFO structure, and is filled in by the SIP manager **58** prior to calling calling GetInfo( ). The hImageNarrow field is a handle to an image-list containing narrow (16×16) masked bitmaps for the Input Method **64**. Similarly, hImageWide is a handle to the image-list containing wide (32×16) masked bitmaps. The SIP manager **58** displays one of the bitmaps (e.g., on the taskbar **56**) to indicate the Input Method **64** that is currently selected. Note that the SIP manager **58** may use the 16×16 or 32×16 bitmaps at various times depending on how it wishes to display the bitmap.

The iNarrow field is an index into the hImageNarrow imagelist indicating which bitmap of several possible from that (narrow) imagelist should currently be displayed. Similarly, the iwide field is an index into the hImageWide image-list indicating which bitmap from that (wide) image list should currently be displayed. Note that the Input Method **64** can initiate a change of the bitmap displayed in the SIP taskbar button **52** by calling IIMCallback::SetImages (described below).

The fdwFlags field indicates the visible, docked and locked states (SIPF_ON SIPF_DOCKED and SIPF_LOCKED) of the Input Method **64**, as well as any special Input Method flags that may be defined in the future. Note that the SIP state flags are ignored for the GetInfo( ) method, but are used in the SetImInfo callback method as described below.

Lastly, the rcSipRect field describes the size and placement of the SIP rectangle. The sizing and placement information returned from GetInfo( ) may be used by the SIP when determining an initial default size and placement. When used, the SetImInfo callback method (described below) specifies the new size and placement of the SIP window **50**.

The ReceiveSipInfo( ) method provides information to the Input Method **64** about the SIP window, including the current size, placement and docked status thereof. This call is made whenever the user, an application **29** or the Input Method **64** changes the SIP state. When the SIP manager **58** sends this information during Input Method initialization, the SIP manger **58** is informing the Input Method **64** of the default SIP settings. The Input Method **64** can choose to ignore these defaults, however the values given are ones that either the user has selected or values that have been recommended as expected or accepted SIP values for that platform. A pointer to the SIPINFO structure that includes this information is passed with this call.

The RegisterCallback method is provided by the SIP manager **58** to pass a callback interface pointer to the Input Method **64**. In other words, the RegisterCallback method call passes an IIMCallback interface pointer as a parameter to the Input Method **64**, whereby the Input Method **64** can call methods on this interface to send information back to the SIP manager **58** as described below. The Input Method **64** uses the callback interface pointer to send keystrokes to applications **29** via the SIP manager **58** and to change its SIP taskbar button icons **52**.

The GetImData( ) method is called when an application program **29** has asked the SIP for the SIPINFOdata structure and has provided a non-NULL pointer for the pvImData member of the SIPINFO structure. The application **29** will ordinarily cause this call to be made when requesting some special information from the Input Method **64**. Two parameters are passed with this call, dwsize, the size of the buffer pointed to by pvImData, and pvImData, a void pointer to a block of data in the application **29**.

With this call, the application **29** is essentially requesting that the Input Method **64** fill the block with information, wherein the size and format of the data are defined by the Input Method **64**. This call is designed for Input Methods **64** that wish to provide enhanced functionality or information to applications. By way of example, a SIP-aware application may wish to know whether a character was entered by way of the SIP or by some other means. An input method **64** can thus respond to the application's request by filling the block.

The SetImData( ) method is called when an application **29** has set the SIPINFO data structure and has provided a non-NULL pointer for the pvImData member of the SIPINFO structure. The application **29** will ordinarily cause this call to be made when requesting that the Input Method **64** set some data therein. The parameters passed with this call include dwsize, the size of the buffer pointed to by pvImData, and pvImData, a void pointer to a block of data in the application **64**.

The IIMCallback Interface

The Input Method **64** uses the IIMCallback interface to call methods in the SIP manager **58**, primarily to send keystrokes to the current application or to change the icon that the taskbar **56** is displaying in the SIP button **52**. The Input Method **64** ordinarily calls the IIMCallback methods only in response to a call thereto which was received through an IInputMethod method call. In general, if the function succeeds, the return value will be a success HRESULT, while conversely, if the function fails, the return value is a failure HRESULT.

The following table represents the IIMCallback Interface:

```
Interface IIMCallback :
Iunknown
{
    Hresult SetImInfo(
            IMINFO *pimi );
    Hresult SendVirtualKey (
            BYTE bVk,
            DWORD dwFlags );
    Hresult SendCharEvents(
            UINT uVk,
            UINT uKeyFlags,
            UINT uChars,
            UINT *puShift,
            UINT *puChars );
    Hresult SendString(
            BSTR ptrzStr,
            DWORD dwChars );
}
```

The first callback, SetImInfo( ) is called by the Input Method **64** to change the bitmaps shown on the SIP taskbar

button **52** representing the current SIP, or to change the visible/hidden state of the SIP window **50**. It is also sent by the Input Method **64** to the SIP manager **58** as a notification when the Input Method **64** has changed the size, placement or docked status of the SIP window **50**. By this mechanism, the various Input Methods **64** are able to alert the SIP manager **58** to these types of changes so that the two remain synchronized. By way of example, an Input Method **64** may wish to have a user interface element which allows the user to toggle between a docked state and a floating state, or between one or more subpanels (e.g. keyboard with buttons to switch to a number and/or symbol panel or international symbol panel). The Input Method **64** uses this call to inform the SIP manager **58** of each change in state.

Although not necessary to the invention, all values passed in the IMINFO structure are used by the SIP manager **58**. Consequently, the Input Method **64** should first determine the current state of the SIP window **50** as provided by the SIP manager **58** in the SIPINFO structure received via a prior ReceiveSipInfo( ) method call, described above. Then, the Input Method **64** should make changes to only those settings in which a change is desired, and pass a full set of values back in the IMINFO structure. The pimi parameter is sent as a pointer to an IMINFO structure representing the new Input Method **64** settings, including the size, placement and state of the SIP window **50** as well as the desired Input Method **64** images.

In response to the SetImInfo( ) call, the SIP manager **58** will show or hide the SIP window **50** as specified in the fdwFlags of the IMINFO structure. However, the SIP manager **58** will not resize or move the SIP window **50** if requested, but will instead update the size and placement information returned to applications **29** when queried. If the specified values represent a change from the current SIP state, the SIP manager **58** will notify applications **29** that the SIP state has changed via a WM_SETTINGCHANGE message, described above.

The SendVirtualKey( ) callback is used by an Input Method **64** to simulate a keystroke for a virtual key, e.g., a character or the like entered via the touch screen display **32** or some other Input Method **64**. The key event will be sent to the window which currently has focus (i.e., the window which would have received keyboard input had a key been pressed on an external keyboard). The SendVirtualKey callback modifies the global key state for the virtual key sent, whereby, for example, an Input Method **64** can use this function to send SHIFT, CONTROL, and ALT key-up and key-down events, which will be retrieved correctly when the application **29** calls the GetKeyState( ) API. The SendVirtualKey callback should be used to send virtual key events that do not have associated characters (i.e., keys that do not cause a WM_CHAR sent as a result of TranslateMessage. Note that WM_CHAR, TranslateMessage and other key-related messages are described in the reference "Programming Windows 95", Charles Petzold, supra). If character-producing virtual keys are sent via this function, they will be modified by the global key state. For example, a virtual key of VK_5 that is sent when the shift state is down will result in a '%' WM_CHAR message for certain keyboard layouts.

Parameters sent with this callback include bVk, which is the virtual keycode of the key to simulate, and dwFlags. The dwFlags may be a combination of a SIPKEY_KEYUP flag, (used to generate either a WM_KEYUP or WM_KEYDOWN), a SIPKEY_SILENT flag, (the key press will not make a keyboard click even if clicks are enabled on the device), or zero.

The SendCharEvent callback allows an Input Method **64** to send Unicode characters to the window having focus, while also determining what WM_KEYDOWN and WM_KEYUP messages the application **29** should receive. This allows the Input Method **64** to determine its own keyboard layout, as it can associate any virtual key with any characters and key state. In keeping with one aspect of the invention, applications **29** thus see keys as if they were sent from a keyboard (i.e., they get WM_KEYDOWN, WM_CHAR, and WM_KEYUP messages). Thus, unlike the SendVirtualKey( ) function, this function does not affect the global key state. By way of example, with the SendCharEvent callback, the Input Method **64** can determine that the shifted (virtual key) VK_C actually sent the Unicode character 0x5564. The shift state flag (specified in the puShift parameter, described below) that is associated with the first character to be sent determines whether a WM_KEYDOWN or WM_KEYUP is generated.

Parameters include uVk, the virtual keycode sent in the WM_KEYUP or WM_KEYDOWN message generated as a result of this function, and a uKeyFlags parameter, a set of KEY state flags that are translated into the lKEYData parameter received in the WM_CHAR, WM_KEYUP or WM_KEYDOWN messages received by the application **29** as a result of this call. Only the KeyStateDownFlag, KeyStatePrevDownFlag, and KeyStateAnyAltFlag key state flags are translated into the resulting lKeyData parameter. The uChars parameter represents the number of characters corresponding to this key event, while the puShift parameter is a pointer to a buffer containing the corresponding KEY_STATE_FLAGS for each character to be sent. If the KeyStateDownFlag bit is sent, this function generates a WM_KEYDOWN message, otherwise it generates a WM_KEYUP message. Lastly, the puchars parameter is a pointer to a buffer containing the characters to be sent.

An Input Method **64** may use the SendString callback to send an entire string to the window which currently has the focus, whereby a series of WM_CHAR messages are posted to the application **29**. An Input Method **64** would typically use this callback after it has determined an entire word or sentence has been entered. For example, a handwriting recognizer or speech recognizer Input Method **64** will use the SendString callback after it has determined that a full word or sentence has been entered.

Parameters of the SendString callback include ptszStr, a pointer to a string buffer containing the string to send, and dwSize, the number of characters to send. This number does not include the null-terminator, which will not be sent.

As can be seen from the foregoing detailed description, there is provided an improved method system for entering user data into a computer system. The method and system are both efficient and flexible, and function with touch-sensitive input mechanisms. With the system and method, a plurality of applications can receive user input from a common input method, while interchangeable input methods may be selected from among a set thereof for each application. The method and system are cost-effective, reliable, extensible and simple to implement.

While the invention is susceptible to various modifications and alternative constructions, a certain illustrated embodiment thereof is shown in the drawings and has been described above in detail. It should be understood, however, that there is no intention to limit the invention to the specific form disclosed, but on the contrary, the intention is to cover all modifications, alternative constructions, and equivalents falling within the spirit and scope of the invention.

What is claimed is:

1. In a computing environment, a computer-implemented method comprising:

displaying an actuatable icon representative of an input method list that includes one or more selectable input methods for one or more computer programs, wherein each input method is a computer-executable software component distinct from the computer programs;

in response to actuation of the actuatable icon, displaying the input method list;

receiving a selection of an input method from the input method list;

installing an input method component that corresponds to the selected input method, the input method component causing an interactive input panel to be displayed;

receiving input via the interactive input panel; and

providing the input to a computer program of the one or more computer programs as if the information was received via user input received from a hardware input device.

2. The method of claim 1 wherein providing the input to the computer program comprises communicating information representative of the input to a graphical windowing environment.

3. The method of claim 2 wherein communicating the information comprises passing the information to an interface.

4. The method of claim 2 further comprising, communicating the information from the graphical windowing environment to an application program, wherein the computer program includes the application program, wherein the information is provided to the application program in a same manner as if the input was received via a hardware keyboard.

5. The method of claim 2 wherein providing the input to the computer program comprises placing the information, by the graphical windowing environment, in a message in a message queue of the computer program, wherein the message queue capable to receive messages corresponding to input from the selected input method and messages corresponding to input from a hardware input device.

6. The method of claim 1 wherein the selected input method corresponds to a displayed keyboard, and wherein receiving input via the interactive input panel that corresponds to the selected input method comprises receiving information corresponding to a keyboard character entered via the displayed keyboard.

7. The method of claim 1 wherein the selected input method corresponds to a handwriting input area, and wherein receiving input via the interactive input panel that corresponds to the selected input method comprises receiving information corresponding to handwritten data.

8. The method of claim 1 further comprising, hiding the input panel.

9. The method of claim 1 further comprising, docking the input panel.

10. At least one computer-readable medium having computer-executable instructions, which when executed perform the method of claim 1.

11. At least one computer-readable medium having computer-executable instructions stored thereon, which when executed by a computer system perform steps, comprising:

selecting one of a plurality of executable input methods for supplying user input to the computer system, wherein each executable input method is an interchangeable software component distinct from one or more application programs, each executable input method having a defined interface set such that the executable input method is connectable to the application programs;

opening an input window on a display of the computer system independent of a window of an active application program; and

displaying an interactive input panel in the input window, the interactive input panel corresponding to the selected executable input method such that information corresponding to user input received by the selected executable input method via the interactive input panel is provided to the active application program as if the information was received via user input at a hardware input device.

12. The computer-readable medium of claim 11 further comprising, providing an input panel button on the display of the computer system, the input panel button being responsive to open and to close the input window.

13. The computer-readable medium of claim 11 further comprising, providing a Software Input Panel (SIP) menu button on the display of the computer system, the SIP menu button being actuatable to display a selectable list of the plurality of executable input methods.

14. The computer-readable medium of claim 13 further comprising, receiving a selection of one of the plurality of executable input methods displayed in the list as a selected executable input method, and in response, closing any open input window, and opening a new input window corresponding to the selected executable input method.

15. At least one computer-readable medium having computer-executable instructions, which when executed perform steps, comprising:

presenting icons corresponding to a plurality of input methods available for a computer application, wherein each input method is a computer-executable software component distinct from the computer application;

invoking a selected input method in response to a user selecting an icon corresponding to the selected input method, including presenting an input panel window; and

accepting user data entered in the input panel window for the computer application, wherein the user data is provided to the computer application as if the user data was received from a hardware input device.

16. The computer-readable medium of claim 15 wherein accepting user data includes detecting user interaction with a touch-sensitive display.

17. The computer-readable medium of claim 15 wherein each input method comprises a component object model (COM) object, and wherein the step of invoking the selected input method includes the step of instantiating the COM object.

18. The computer-readable medium of claim 15 further comprising converting the user data to a Unicode character value.

19. In a computing environment, a system comprising,

a manager component stored on one or more computer-readable media and configured:

to manage selection of a selected input method from one or more available stored input methods, wherein each input method is a computer-executable software component distinct from one or more computer programs, and

to send input data corresponding to a user input received at the selected input method to a graphical windowing environment; and

the graphical windowing environment to receive the input data and to send the input data to a computer program of

the one or more computer programs, wherein the input data is sent to the computer program as if the input data was received via user input received from a hardware input device.

20. The system of claim **19** wherein the computer program comprises an application program having focus.

21. The system of claim **19** further comprising an input panel window corresponding to the selected input method.

22. The system of claim **21** wherein the selected input method presents an image representing a keyboard on the input panel window.

23. The system of claim **21** wherein the manager component selectively displays and hides the input panel window.

24. The system of claim **21** wherein interaction with the input panel does not cause the input panel window to receive focus.

25. The system of claim **19** where the input method is displayed on a touch-sensitive display screen.

26. The system of claim **19** wherein the manager component transfers information from the computer program to the selected input method.

27. The system of claim **19** wherein the selected input method calls functions in the manager component via a defined interface set.

28. The system of claim **19** wherein the selected input method comprises an object.

29. The system of claim **19** wherein the selected input method draws an input panel in an input panel window displayed in the graphical windowing environment.

30. The system of claim **29** wherein the manager component selectively displays and hides the display of the input panel window.

31. The system of claim **29** wherein the manager component docks the input panel window.

* * * * *